UCRL-MA-140943-revised

ParaDyn User Manual

ParaDyn: A Parallel Nonlinear Explicit, Three-Dimensional Finite-Element Code for Solid and Structural Mechanics

Carol G. Hoover, Anthony J. De Groot, Robert J. Sherwood

Methods Development Group
Mechanical Engineering

ParaDyn Versions 2.1/4.1
January 2004

ABSTRACT

ParaDyn is a parallel version of the DYNA3D computer program, a three-dimensional explicit finite-element program for analyzing the dynamic response of solids and structures. The ParaDyn program has been used as a production tool for several years for analyzing problems which range in size from a few tens of thousands of elements to several million elements. ParaDyn runs on parallel computers provided by the Department of Energy Advanced Simulation and Computing (ASC) Program, the Department of Defense High Performance Computing and Modernization Program, and the Atomic Weapons Establishment in the UK. In addition to these massively parallel computers, ParaDyn has recently been installed on several Linux cluster computers.

Preprocessing and post-processing software utilities and tools are designed to facilitate the generation of partitioned domains for processors on a massively parallel computer and the visualization of both resultant data and boundary data generated in a parallel simulation. This manual provides a brief overview of the parallel implementation; describes techniques for running the ParaDyn program, tools and utilities; and provides examples of parallel simulations.

PREFACE

Preface to Version 2.1/4.1

Message-passing versions of contact types 3, 5, 8, 9, and 10 are new in ParaDyn Version 4.1. These algorithms augment the local contact algorithms (types 1, 2, 3, 5, 6, 7, 8, 9, and 10) which place each contact interface into one processor. The new message-passing versions of local contact remove the limitation on processor count encountered when the largest local contact surface prevented further partitioning for more processors.

The significant milestone for Version 2.1 of ParaDyn is the release of highly optimized parallel automatic contact algorithms. These parallel automatic contact algorithms incorporate dynamic load balancing and include penalty contact, Lagrange contact and SAND (material erosion) algorithms.

The parallel algorithms in Versions 2.1 and 4.1 were designed for ParaDyn and DynaPart by Tony De Groot and developed by De Groot in ParaDyn and Bob Sherwood in the set of programs run by DynaPart.

The heightened interest and yearly assessments in Software Quality Assurance Processes and Practices have influenced our activities in this area. Software configuration management practices now include a split of the source code into two branches. The production version is characterized by carefully controlled changes to the source, limited to error corrections. The development version is the code-development source and incorporates new algorithms and modifications to existing capabilities. The development version plus alpha and beta testing then lead to a new production version. The version labels in this document refer to the production version of the ParaDyn software. The first two numbers in the version label represent the year of the release, measured from year 2000, and the number of the release within that year. Version 2.1 was released in 2002 and is the first (and only) release made in 2002.

Software Testing with defined alpha and beta testing periods was applied to Version 4.1, released in early 2004.

Analysts contribute significantly to the specification of new requirements, alpha/beta testing, and collaborating on the reporting and corrections of errors. We especially thank analysts in the Advanced Engineering Analysis Group at Lawrence Livermore National Laboratory (LLNL) and

Ed Kokko at Livermore. Neil Hodge from the AEAG has provided the first beta testing of the VisIt parallel visualization tool. Thanks very much Neil! We also thank all of our collaborators, the Engineering Analysis Group at Los Alamos National Laboratory (Scott Doebbling and Bob Stevens) and Photios Papados and Jim O'Daniel, at the Department of Defense Engineer Research and Development Center (ERDC).

Preface to Version 1.0

This User Manual documents the collaborative code development work with my colleagues, Tony De Groot and Bob Sherwood. Our efforts represent the majority of the original work in the parallel algorithm design and development for the ParaDyn Project. Doug Speck, Elsie Pierce, and Vic Castillo are now substantive contributors to the ParaDyn Project. Doug and Elsie, in particular, have made it possible to quickly design flexible binary databases (Mili) and have developed significant enhancements to the GRIZ visualization software. Their work paves the way for the visualization capabilities needed when using parallel computers with hundreds and thousands of processors.

Analysts provide the insights needed to turn our software development into an effective production tool for finite-element engineering analysis. At the Lawrence Livermore National Laboratory (LLNL) Dan Badders, Tony Lee, and Tony DePiero entered the turbulent waters of massively parallel computers earliest and have provided very valued input to our code development efforts.

Raju Namburu and Photios Papados are collaborators from the Army Research Laboratory (ARL) and the Engineer Research and Development Center (ERDC). They led the way to the first multimillion element calculations on Department of Defense high performance computers. John Benner and colleagues from the Los Alamos National Laboratory (LANL) were the first to use the ParaDyn program on over 1000 processors.

We especially appreciate the very valuable comments and support that our collaborators from ARL, ERDC, and LANL have provided for our code development efforts.

Carol Hoover
Methods Development Group
Lawrence Livermore National Laboratory
Livermore, California 94550
925-422-1556 hoover1@llnl.gov

TABLE OF CONTENTS

1.0 BACKGROUND

Significant speed gains (for example, factors of sixty or more on sixty-four processors) are being achieved for engineering design calculations on parallel computers with as few as a hundred processors using ParaDyn, the parallel version of the DYNA3D program [1]. The latest massively parallel computers with thousands of processors now make it possible to design engineering models for mechanical system analysis with multiple component parts as well as to add more detail and complexity in the models for components. ParaDyn and DYNA3D are explicit finite-element programs designed to solve for the nonlinear, transient response of solids and structures. The two programs, contained within a single source, are developed by the Methods Development Group at the Lawrence Livermore National Laboratory (LLNL). The web site, http://www.llnl.gov/eng/mdg/mdg_home.html, provides general information and online documentation about the complete family of thermomechanical programs developed in the Methods Development Group.

Parallel computers are now commonplace in our computing environment. Computers with speeds ranging from 1 to 100 TeraOps (10^{12} to 10^{14} operations per second) and thousands of processors will be delivered through the year 2005 to the Advanced Simulation and Computing (ASC) Program. These computers dominate the high-end computing at LLNL. Complementing these resources, the Livermore Laboratory is making a significant investment in distributed-memory Linux clusters. These systems are characterized by off-the shelf processors (such as the Intel Xeon processors) with an interconnect network. The performance of ParaDyn on these systems depends very much on the balance between processor speeds and the speed of the interconnect network. A summary of the computer services and resources at LLNL may be viewed at http://www.llnl.gov/icc/lc.

The Department of Defense High Performance Computing and Modernization Program is also a very strong participant in acquiring hardware and developing software for next-generation massively parallel computers. Computer characteristics in terms of size, speed, and number of processors at the DOD Major Shared Resource Centers (MSRCs) are summarized in the links provided by the Program Office web site at http://www.hpcmo.hpc.mil. These data illustrate the continuing significant increase in our parallel computational speeds and capacities.

ParaDyn is a parallel production program. Code development efforts are now directed toward optimizing the parallel algorithms, developing parallel preprocessing and post-processing software, and developing software tools for engineering optimization studies. Concurrent with the parallel development effort, many code developers for the DYNA3D program are contributing new algorithms, elements, and material models to enhance the mechanics modeling capabilities. A single source encompassing both the ParaDyn and DYNA3D algorithms makes it possible to easily migrate the DYNA3D enhancements into the production ParaDyn program. Finally, coupled programs for thermal, mechanical, and fluid analysis are being designed for parallel computers. A production capability for coupled analysis is possible in the next few years.

2.0 OVERVIEW OF PARADYN

2.1 Introduction

Advances in the development of parallel algorithms for explicit finite-element analysis and domain partitioning techniques have led to scalable production applications using ParaDyn. This has resulted in several benefits to our engineering design programs. Firstly, calculations are now performed in a day or less for problems that previously ran over several weeks. Secondly, new models are being generated for mesh sizes between one-million and ten-million elements. This is an order of magnitude larger than the largest models possible in the past. Finally, longer time simulations (problems running for a few million steps) are now being run on both massively parallel computers and Linux clusters.

Analysts play an important role in preparing models for parallel computers. The meshes are increasingly much larger and more complex. New validation tools are needed for the mesh generation step, specification of boundary loads and constraints, and defining facets on interfaces. In addition, the modeling of contact interfaces can have a significant effect on the parallel performance and scalability. Optimizing the performance and achieving scalability of parallel contact algorithms is particularly challenging and has been the focus of algorithm research in ParaDyn in the last several years. An overview of the parallel contact algorithms is presented in Section 2.4.

ParaDyn is a production program and includes a suite of software for automating the preparation of models and the analysis of results. The DynaPart preprocessing tool automates the task of model partitioning and is coupled directly to the research in scalable parallel contact algorithms developed in the ParaDyn program. The development of the GRIZ4 visualization tool [2] based on the Mili (Mesh I/O Library) software [3] provides a flexible, self-defining format to use for the large databases generated during a parallel simulation. Another recent advance which provides additional analysis tools has been the development of routines for reading families of Mili databases generated in parallel simulations. These routines have been implemented in VisIt, the parallel visualization tool developed by the ASC Program at Livermore and also in EnSight, a commercial parallel visualization tool supported by the ASC Program and used at Los Alamos.

The ParaDyn software suite consists of the following software products:

- DynaPart, a partitioning tool for the spatial decomposition of the model input data;
- DYNA3D, a nonlinear, explicit, three-dimensional finite-element code for modeling the dynamics of solids and structures;
- ParaDyn, the parallel version of the DYNA3D program;
- Mili, an Input/Output (I/O) subroutine library for formatting binary databases used in finite-element computer programs;
- Xmilics, a tool for combining and splitting families of Mili databases created on a parallel computer;
- GRIZ4, a visualization tool for post-processing results.

There are two additional parallel visualization tools that have been used for visualizing results in Mili databases from ParaDyn. These are:

- VisIt, a parallel visualization tool for post-processing results.
- EnSight, a commercial parallel visualization tool used by analysts at Los Alamos.

VisIt is currently undergoing beta testing with the Mili databases generated on parallel computers at LLNL.

For installations where the ParaDyn software suite is available, all of the documentation is in PDF form and stored in directories with the executables and related libraries. Other individuals or groups interested in the capabilities of the software developed by the Methods Development Group (MDG) can access documentation on the MDG Home page. The Home page web address is http://www.llnl.gov/eng/mdg/mdg_home.html. The VisIt software includes PDF documentation and instructions for downloading the software. The Home page for VisIt is http://www.llnl.gov/visit.

To prepare input data and select control options and flags for the ParaDyn program, follow the discussions in the DYNA3D User Manual [1]. Use Section 3 in this manual to follow the steps needed to partition a model, run the problem on a parallel computer, and use the post-processing tools for analyzing the results. Instructions for using the DynaPart preprocessing software are outlined in Section 3 and discussed in depth in the Appendices. The Xmilics tool is discussed in Section 3 and a Help package is displayed interactively by using the execute line with no arguments. The post-processing software is documented in the GRIZ4 User Manual. Finally, some standard features in DYNA3D requiring additional documentation for a parallel analysis are included in Section 4 of this manual.

A set of typeface conventions is followed throughout this manual to allow the reader to easily distinguish between **commands**, *parameters*, and `computer generated text`. **Commands** that appear in **bold** type should be entered verbatim. *Parameters* that appear in *italic* type should be given values when included in the input. `Computer generated text`, such as error messages or default file names, is printed in a `typewriter-like (Courier)` font. In text passages file names appear in italic type for clarity.

The next sections provide introductory discussions about parallel algorithms and computers. Section 2.2 discusses the parallel finite-element model and describes partitioning methods. Section 2.3 discusses parallel performance and scalability measurements. Section 2.4 characterizes contact interfaces and their implementation in parallel. Section 2.5 lists boundary conditions and other options which are given special treatment by the partitioning software. Section 2.6 discusses scalability studies for different models and parallel computers.

2.2 The Parallel Finite-Element Model

A successful strategy for parallel implementation of the explicit finite-element method is based on dividing the mesh among the processors and executing ParaDyn on a subdomain in each processor [5]. The elements from the mesh are divided into subdomains so that each processor has approximately the same amount of calculations to perform in a timestep. The nodes on the boundaries of a subdomain are referred to as shared nodes. Nodal force data for shared nodes are communicated between processors when the nodal force updates are calculated. The nodal points on the subdomain boundaries areduplicated (shared) in more than one processor. Mesh partitioning is the strategy for dividing the problem into subdomains and mapping subdomains to processors. This is illustrated for two processors in Figure 1.

The nodal forces consist of contributions from applied loads, contact interactions, and internal deformations.

$$\mathbf{F}_{\text{node}} = \mathbf{F}_{\text{applied}} + \mathbf{F}_{\text{contact}} - \mathbf{F}_{\text{internal}}$$

The internal force calculation for a shared node includes a contribution from elements in different processors. Each processor calculates a partial nodal force for the elements in its processor. These partial force contributions are communicated between the processors so that the total force computed for a shared node is the same in all processors within the error introduced by the ordering of the calculations. As much as possible, the ParaDyn algorithms are designed to store partial nodal force data for shared nodes until all contributions to the nodal force have been computed before communicating the shared data.

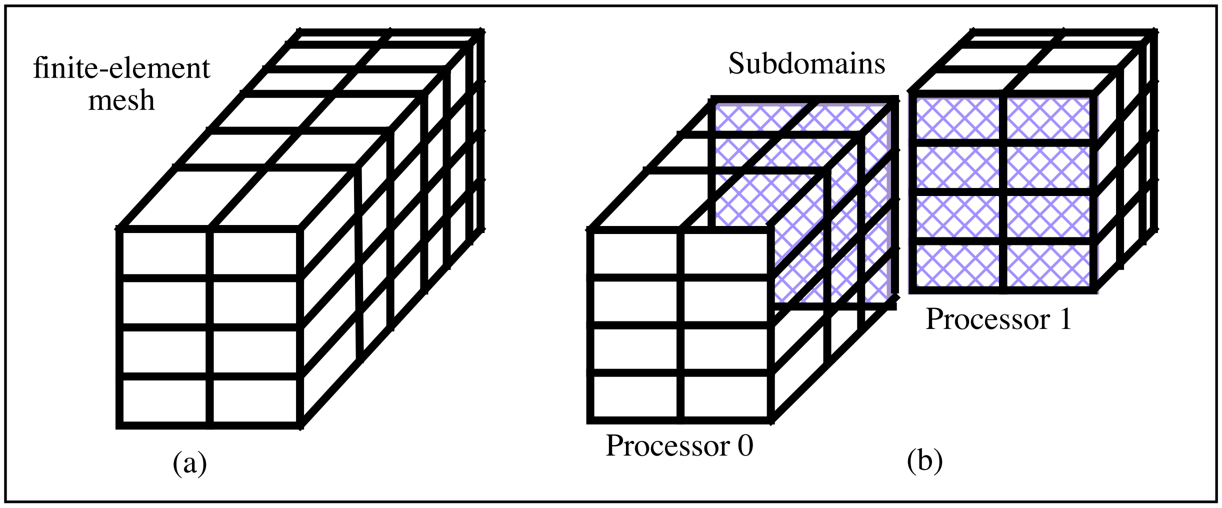
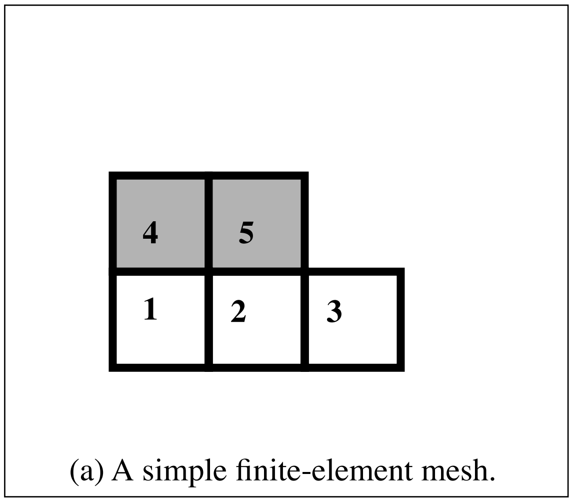


Figure 1. Two subdomains on a finite-element mesh. (a) The original mesh with 48 elements is partitioned into two subdomains with 24 elements each. (b) The calculations involving nodal points on the cut plane (shown as patterned) are performed in both processors. The 15 nodes on the cut plane are referred to as shared nodes.

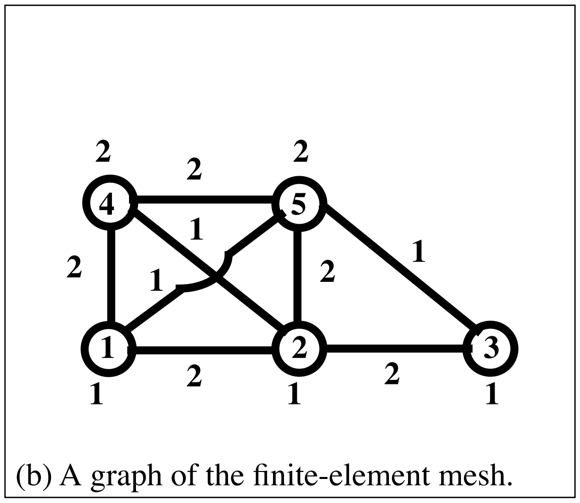
Research in applied mathematics has led to efficient techniques for subdividing or partitioning the complicated unstructured meshes that arise in practical engineering applications [6-9]. We use the METIS software from the University of Minnesota to partition finite-element meshes and contact surfaces. (For more information on METIS, see http://www-users.cs.umn.edu/~karypis/metis/main.shtml). The METIS algorithms use a graph to represent the finite-element mesh. Preprocessing software automatically produces the graphs needed for the mesh partitioning step.

Mesh partitioning is accomplished by representing a finite-element mesh as a graph. A graph has vertices and interconnecting edges. The vertices and edges represent objects on the mesh. For finite-element meshes, the vertices of the graph correspond to elements (zones) in the mesh and the edges correspond to nodes in common between two connected elements. See Figure 2.

The graph represents the element-to-element connectivity for the mesh. The METIS algorithms find an efficient division of the graph corresponding to a specified number of subdomains. An important aspect of graph partitioning techniques is the use of weighting factors for both the vertices and edges. These weighting factors are used to balance the vertices into sets of roughly equal weight. This weighting of the graph provides the best representation of both the computational costs and the communication costs for the partitioning of the mesh. To illustrate this, the relative computational cost for a complex material model, a boundary condition or any other expensive part of the calculation, can be used to weight the vertices. Similarly, an edge in the graph represents the number of common nodes between the elements and can be appropriately weighted by a relative measure of the shared data communicated if the graph is cut on that edge. Figure 2.(b) indicates the edge weights used for the few elements shown in Figure 2.(a).



(a) A simple finite-element mesh.



(b) A graph of the finite-element mesh.

Figure 2. A finite-element mesh and the graph representation of the mesh. (a) This simple mesh consists of two materials, shaded and unshaded. The shaded material requires twice as much calculation time as the unshaded material. (b) This is the graph of the mesh. The vertices are represented as circles and the number near a vertex is the computational weight of the vertex. The lines connecting the vertices are edges and represent the shared data between the vertices. The number specified along the edges represents the number of shared nodes between the two elements represented by the vertices.

The partitioning task is automated completely in the preprocessing tool, DynaPart, for any mesh geometry. This software was used for the assignment of a one-million element mesh to 128 processors as shown in Figure 3. The colors are used to show the processor assignment for subdomains on the mesh. The mesh was developed to model a shock moving from the top vertex of the mesh in the direction of the half-cylindrical cavity region located half way down and on the left-hand side of the mesh. The mesh is zoned very finely at the top of the model to resolve the shock structure. As a result of this fine zoning the subdomains are much smaller at the top of the mesh than the subdomains on the lower part of the mesh where the zoning is coarse. The mesh lines are not shown in Figure 3.

.

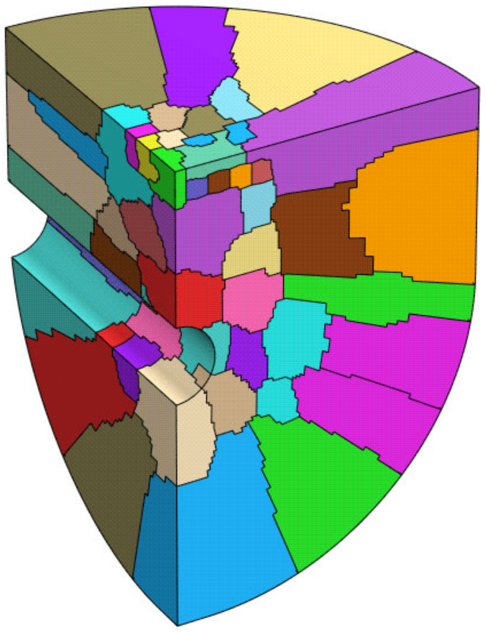


Figure 3. Processor assignment for a one-million element mesh. Colors are used to distinguish the subdomains assigned to 128 processors. This problem without sliding interfaces scales linearly as the number of processors is increased up to roughly 1000 processors.

2.3 Parallel Performance and Scalability

The time required to deliver results on a parallel computer is the sum of the time for computing results on the processors and the time for communicating data between the processors.

$$\tau_{wc} = \tau_{calc} + \tau_{comm}$$

where

τ_{wc} is the total elapsed time taken for the calculation (total wall clock time);

τ_{calc} is the elapsed time the processors spend computing results;

τ_{comm} is the elapsed time the processors spend communicating shared data.

Ideally, if the number of processors is doubled, the rate for delivering results will be doubled. The term *scalability* or *theoretical scalability* refers to this linear scaling of the delivery rate with the number of processors. In practice scalability breaks down when the communication time becomes significant compared to the amount of time processors spend computing results. Speedup curves measure the calculation rate versus the processor count for specific computers. Examples are shown in

If the parallel calculation is efficient, the delivery time τ_{wc}, for N_p processors will be approximately equal to $1/N_p$ of the time for calculating the same results on one processor, τ_1. Thus, a measure of the parallel efficiency is given by the following:

$$\varepsilon = \tau_1/(N_p\tau_{wc}) \times 100\%.$$

Almost all problems of interest include contact interface definitions which are always challenging to run efficiently in parallel. See Section 2.4. One of the software programs in DynaPart (reducegrf) calculates an estimate for the maximum number of processors that can be used for a ParaDyn simulation based solely on balancing the computational work for *both* the element deformation and the contact calculations. Note that interprocessor communication time is ignored in this estimate. Using more processors will result in no further improvement in delivering results and wastes computing resources. On the other hand, an efficient calculation may be limited by the number of processors even further for any specific hardware platform. In this case, hardware speeds, CPU and network are taken into account. Section 2.6 illustrates this with examples.

2.4 Scalable Parallel Contact Algorithms

Designing efficient parallel contact algorithms is challenging for several reasons:

- A load balanced mesh partition may split a contact interface into many processors and potentially cause a large amount of unnecessary communication of contact nodes and facets.
- Communication may be necessary for parallel contact algorithms if the relative distance moved by the two surfaces is more than one element length.
- For problems with moving parts, the material interfaces in contact change dynamically. Search algorithms for finding interfaces in contact are relatively time consuming compared to the contact enforcement or element deformation calculations.

Because of these characteristics of the interface calculations, dividing the problem domain into subdomains that are optimal for calculating the element deformations may not result in an efficient division of the problem for the contact calculations. In some cases the ParaDyn software uses partitioning methods for the sliding interfaces that differ from the partitioning for the mesh. As a result the contact calculations are performed in a different set of processors than the element calculations for connected elements. The contact force results are communicated once in a timestep to the processors defined by the mesh partition. Similarly the nodal position and velocity updates are communicated from the processors defined by the mesh partition to the processors defined by the contact partitions once in a timestep.

There are currently two parallel contact algorithms in the ParaDyn program: *local contact* and *arbitrary contact*. In some problems, surfaces initially close together engage in small relative motion and the contact remains in a localized region of the mesh. We refer to this as *local contact*. Sliding interface types (1-3, 5-10) in DYNA3D/ParaDyn are *local contact* algorithms. For other problems with many components and with large relative motion at the interfaces, the interactions of the surfaces are not predictable. Thus, more sophisticated and time consuming searches for the surfaces in contact must be performed throughout the simulation. We refer to this as *arbitrary contact*. Examples illustrating arbitrary contact are a ball rolling on a plane, a surface folding on itself, material fragmentation and failure, or an automobile crash simulation. Arbitrary contact is implemented in DYNA3D automatic contact interfaces, types 12 through 14. The automatic contact algorithm type 14 is the material erosion algorithm (SAND).

In ParaDyn Version 2.1 the parallel algorithm for local contact places a contact interface (both the master surface and the slave segments or nodes) into one processor. This method is very efficient and useful for problems with many contact surfaces that are relatively small. However, the method can limit the scalability of the problem if there are large contact surfaces because the partitioning for contact may require more elements in a processor than is efficient for an optimal mesh partitioning. ParaDyn Version 4.1 removes this scalability limitation by splitting a local contact interface over a group of processors instead of just one processor, if needed. This new feature applies to local contact algorithms 3, 5, 8, 9, and 10.

The parallel arbitrary (automatic) contact algorithm uses a localization technique (a bucket search) to search for contact points. The partitioning of arbitrary contact surfaces is a step separate from the mesh partitioning. Thus, there are two distinct partitions of the model and there is communication between the two partitions at each timestep. The parallel arbitrary contact algorithm also redistributes the contact surfaces over the processors if the surface motion results in motion over a distance larger than one bucket length. This redistribution step is referred to as *dynamic load balancing*.

Sliding interface type 11 (SAND) is not implemented in parallel because the equivalent interface is modeled in the more robust and newer automatic contact interface type 14. The single-surface contact algorithm in type 4 is implemented as a full surface assigned to a processor. Single surface contact may also be modeled using interface type 13. Type 13 is a newer algorithm with more features and for many problems it is preferable to type 4.

The most commonly used sliding interface in DYNA3D is type 3. The same physical interface condition (sliding with friction and voids) can also be modeled with the automatic contact algorithm types 12 and 13. The parallel efficiency for either choice for the sliding interface may vary significantly depending on the details of the interfaces and the size of the model. The next sections discuss the two forms of parallel contact algorithms.

2.4.1 Parallel Local Contact

In ParaDyn Version 2.1 DynaPart always places each local contact interface in one processor. In ParaDyn Version 4.1 DynaPart will split up local contact interface types 3, 5, 8, 9, and 10 if the largest contact surfaces limit the number of processors that can be used. The DynaPart keyword *localparallel* must be used in ParaDyn Version 4.1 to enable the splitting of the local contact surfaces.

For more than one local contact sliding interface, DynaPart distributes the set of interfaces among as many processors as possible. Special graph weighting methods are used for partitioning the local contact interfaces and evenly distributing the local contact calculations among processors.

A spin forming mesh illustrates the partitioning for local contact without the *localparallel* option and is shown in Figure 4. This model consists of a rotating plate formed into a hemispherical shape by rollers in contact with the plate surface. A four-processor partition for this problem cuts the plate into three concentric rings. The center ring and brushes form the sliding interface and are fully contained in one processor. An eight-processor partition for the problem cuts two of the rings in half. The sliding interface, the center ring and the rollers, remains uncut.

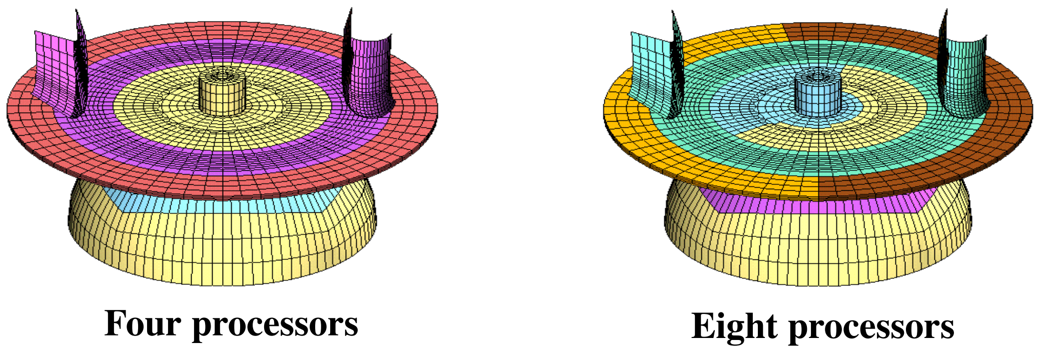


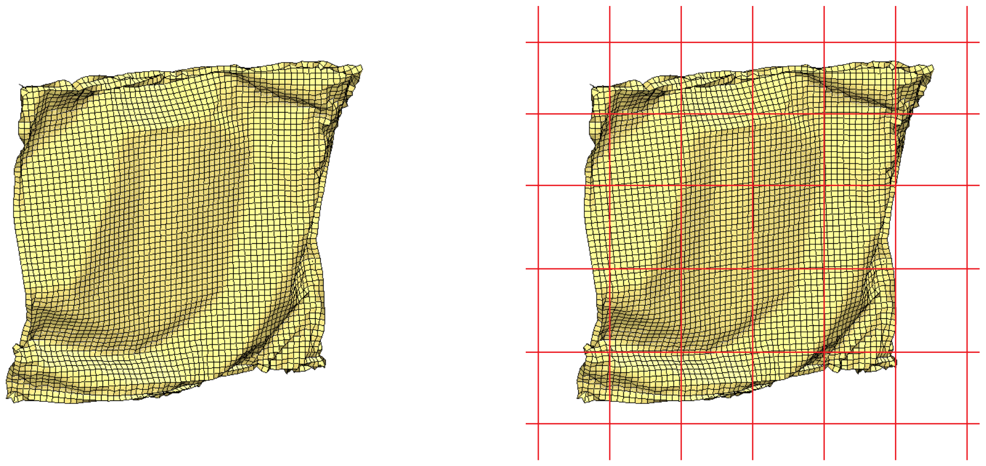
Figure 4. Four and eight-processor partitions for a spin forming problem. The model partitioned for four processors divides the plate into three concentric rings. The eight-processor partition leaves the middle ring uncut.

2.4.2 Parallel Automatic Contact

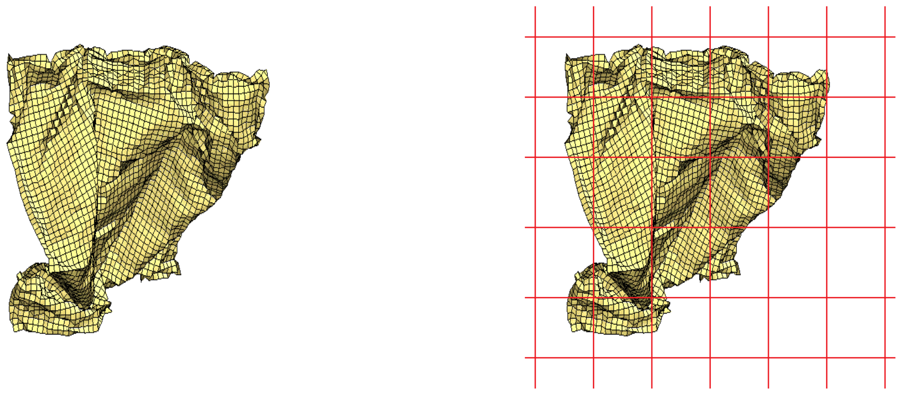
For arbitrary contact defined with sliding interface types 12-14, the analyst avoids the very time consuming task of defining the contact surfaces in the model. However, the parallel algorithm for these interface definitions is more expensive for two reasons. First, the search for contact is an expensive step, both on single processor computers and on parallel computers. Furthermore, for problems with considerable surface motion, the parallel versions of these algorithms require additional data communication because the buckets on processor boundaries can be located in multiple processors. A periodic surface rebucketing is needed (on serial and parallel computers) to avoid missing contact points.

An example of arbitrary contact is shown in Figure 5. This is a material (shells) which is folding as a result of six boundaries (on a cube) moving from the edge of the original square towards the center.

The first step in the arbitrary contact algorithm is a sort step to localize the search for material interfaces. The sort algorithm generates a set of buckets (cells) for grouping surface nodes and facets into localized regions on the mesh. The parallel version of the algorithm partitions the buckets among the processors to balance the contact calculations among them and minimize the communication of nodes and surface facets in buckets with data that must be shared between processors.



a) A folding surface with an overlaid set of buckets for localizing the contact search



b) The folding surface at a later time with a new set of buckets for the contact search

Figure 5. The arbitrary contact algorithm is used for surfaces with unpredictable motion such as this folding sheet of material. a) Buckets are used to localize the search for contact. b) Once the surface moves more than the length of a bucket, the buckets must be regenerated. New buckets may be of a different size and distribution than the original set of buckets.

An additional step is needed in the automatic contact algorithm when the surface motion results in nodes and facets moving more than one bucket width. When this happens it is necessary to resort the surfaces so that contact points are not missed. The frequency of this bucket-regeneration step is automatically computed or, in some unusual situations, can be specified as a user input. In

ParaDyn the bucket-regeneration step also includes a partitioning of the buckets to load balance the new set of buckets among the processors. The bucket regeneration and partitioning require extra communication during the timestep over which it occurs. Thus, it is important to rebucket as infrequently as possible to avoid the extra communication costs.

The efficiency of the parallel arbitrary contact can be improved by limiting the search domains with the DYNA3D keyword input options listed in the section for contact algorithms. Sliding interface types 12 and 13 are identical. The options for controlling the search domain and other features as follows:

- Boxes can be defined to limit the domain of the search;
- Material can be included or excluded in the boxes for the search;
- Faces defined as in a type 3 interface can be specified rather than automatically generated.

Refer to Section 4.1 for a discussion of modeling tips for developing models that result in the most efficient performance from the parallel contact algorithms.

2.4.3 Modeling Tips for Efficient Parallel Contact

Computational performance should increase if all contact surfaces are kept as small as possible. If any contact surface can be defined as two or more independent smaller surfaces, this should be done during mesh generation or with the use of DYNA3D keywords. For efficient parallel arbitrary contact (automatic contact) it is very important to use DYNA3D input options to limit the search domains.

Section 4.1 provides valuable *modeling tips* for all forms of parallel contact. It is very important to read Section 4.1 before designing the contact for models that will be run using ParaDyn.

For a complex mesh it is beneficial to use both local and arbitrary contact algorithms and to provide multiple instances for each type of contact. An obvious advantage in doing this is that the regions on a mesh without any contact are not included in either the partitioning for local contact or the search domains for arbitrary contact.

2.5 Boundary Conditions and Constraints

Parallel versions of boundary conditions and constraints are treated both in the partitioning software as well as in ParaDyn. Because the partitioning software uses special processing on selected boundary conditions and constraints, it is important to know which boundary conditions and constraints are treated with partitioning and how this affects the overall partitioning.

The following DYNA3D options affect the partitioning by placing all associated nodes and elements into one processor. The largest set of nodes for any of these features will limit the maximum number of processors that can be used efficiently. It is therefore best to keep the size of each set of these boundary conditions and constraints as small as possible.

Symmetry planes with failure
Follower forces
Nodal constraints
Sliding interface definitions:
  Types 1 2 4 6 7 in Version 4.1
  Types 1 through 10 for Version 2.1
Tie-breaking shell slidelines
Tied node sets with failure
Rigid body joints
Shell-solid interfaces
Discrete springs and dampers
One-dimensional slidelines.

These DYNA3D objects contain nodes and elements that must be assigned to a single processor rather than divided across more than one processor. Nodes that need to be kept together are assigned to Special Nodal Points (SNP) sets in the partitioning software. Associated with each of the SNP sets is a Special Element (SE) set. The SE set consists of all elements that contain one or more nodes in the corresponding SNP set. Each SNP set and its associated SE set must be fully contained in a processor. As a result, large SNP or SE sets can constrain a mesh partitioning and limit the number of processors that can be used for the problem.

2.6 Testing and Evaluating Model Scalability

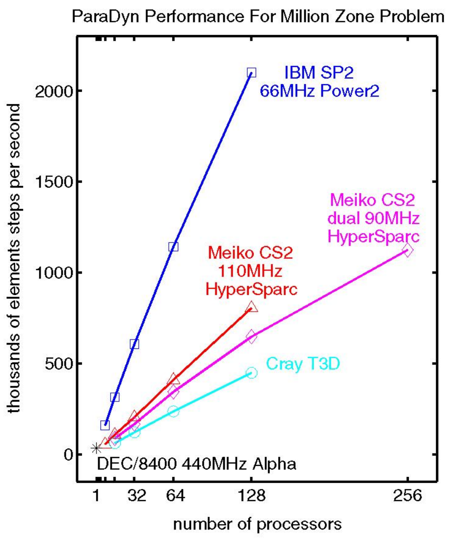
If processors are readily available on the parallel computer, it is a good idea to partition the model more than once in order to select an optimal number of processors to use for the simulation. Selecting an optimal number of processors means you use as many processors as you can (and thus, get the best turnaround for your results) and at the same time you run the parallel simulation efficiently. The partitioning software provides statistics that allow you to make a reasonable estimate for the number of processors to use for achieving a computational load balance. On the other hand, it is not possible at the partitioning step to show that the communication time is negligible compared to the calculation time.

Therefore, it is necessary to benchmark the ParaDyn performance with scalability studies for all parallel machines that will be used and for the prototype models of interest. This step is not necessary if other analysts have done this study already.

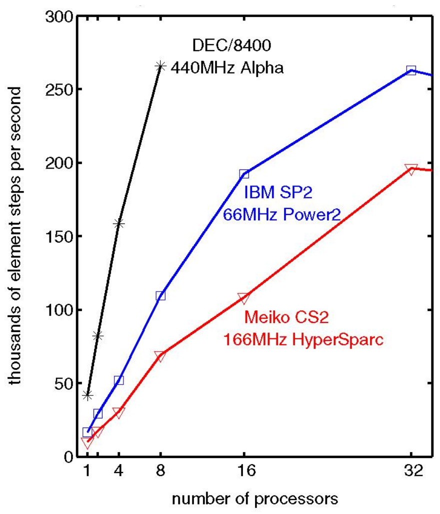
Tests using the current systems at LLNL have shown that processing speed increases with the number of processors until the number of elements per processor drops below approximately 2000 to 4000. This estimate may be inadequate if contact surfaces, boundary conditions, or nodal constraints are restricting the partitioning. The reason is that the optimization of the parallel algorithms for these options constrains the nodal data to reside on one or a small number of processors. See discussions in Sections 2.4 and 2.5. Finally, it is important to generate enough points on the speedup curve to determine the number of processors at which the speedup curve begins to deviate from the ideal linear speed up.

2.6.1 Speedup Studies and Scalability Limit

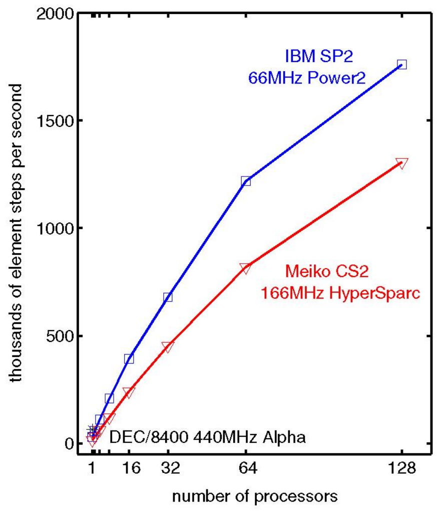
Speedup studies and the scalability limit are important to understand for any computer platform on which ParaDyn is used. To generate a speedup curve, the wall clock time is measured as a function of the number of processors. Then the rate of calculation is plotted as a function of the number of processors. Perfect scalability or *theoretical scalability* means that the calculation rate is doubled as the number of processors is doubled. Thus a perfectly scalable calculation is one with a straight line at 45 degrees when the calculation rate is plotted as a function of processor count. In practical problems a straight line behavior with a slope less than the theoretical scalability is sufficient and usually indicates that communication time is affecting the rate at which processors receive data needed to complete calculations for nodal data shared between processors.



a) Parallel speedup for a million-element problem with no contact surfaces



b) Parallel speedup for a 50,000-element model with 27 contact surfaces



c) Parallel speedup for 100,000-element model with 30 contact surfaces and 20,000 surface nodes

Figure 6. The speedup curves shown here are for older parallel computers, but the speedup characteristics show that the balance between the CPU speeds and the interconnect speeds are quite relevant to the performance of ParaDyn. a) This problem has a small percentage of the nodes being communicated between processors. The curves stop at the maximum number of processors available on the computer. Typically ParaDyn was run with the largest number of processors available on these machines. b) These speedup curves show that load balance can limit performance. The problem was run with 8 or 16 processors. c) A similar degradation in load balance occurs here. The problem was run on 64 processors.

In ParaDyn timing values are printed in the standard output file (d3hsp) each time the problem energy statistics are printed. These statistics are useful because they provide an indication of the extra cost of a contact rebucketing timestep compared to a step without rebucketing. For the speedup curves the final timing statistics measured for the problem and printed at the end of the standard output file should be used to compute the calculation rate. This value at the end of the run amortizes the differences in the time for timesteps with and without rebucketing for contact. Speedup curves for problems with and without contact are illustrated in Figure 6.

The processor count at which there is a deviation from linearity in the speedup curve is usually selected as the maximum number of processors for which the problem is scalable. For ParaDyn this point on the speedup curve is often characterized by large and unpredictable communication times. DynaPart programs collect statistics about the model objects that require splitting to maintain good load balance and the statistics on the cut edges for each partition. From these data DynaPart estimates and prints the processor count at the scalability limit and labels it “Max # partitions” (See discussion in Section 2.6.2). This processor count is an estimate of the scalability limit to load balance the calculations. It does not account for the hardware communication times and the balance between the hardware CPU and interconnect communication times. It is for this reason that the speedup curves should be studied for each platform on which ParaDyn is used.

2.6.2 Using Statistics from the Partitioning Software

After gaining an understanding of the performance characteristics of ParaDyn for specific parallel platforms and prototypical models, it is possible to use statistics in the DynaPart log file as a guide for selecting the number of processors.

Computational Load Balance:
The **skmetis** program in DynaPart calculates and prints statistics for evaluating the quality of the partitioning with respect to the computational load balance among the processors. This output is listed as the *Load balance* number from METIS. The best balance is obtained when this number is as close as possible to a value of 1.00. The *parallel speedup* is given by the ratio of the number of processors to the load balance. This assumes the communication time is negligible and the deviation of the speedup from the ideal is due to the imbalance in the calculations across processors.

The UNIX **grep** command shown below prints the line in the DynaPart log file containing the computational load balance statistic.

grep "Load balance:" *logfilename*

```
Edge cuts:    1080,  Load balance:  1.00000,  Score: 1.55556.
```

Maximum Number of Processors Limited by Contact Surfaces and Other Model Options:

The partitioning of models particularly with slide surfaces may limit the number of processors that can be used for an efficient ParaDyn simulation. See Sections 2.4 and 2.5. The DynaPart software computes and prints a good estimate for the optimal number of processors to achieve computational balance when these options are used in the model. The model can then be repartitioned using the *minimum* of this value and the optimal the number of processors found on the speedup curve. Use **grep** to get this statistic from the DynaPart log file as follows:

grep "Max # partitions" *logfilename*

```
Max # partitions for good load balance:       27
```

Balancing and Limiting Communication Time:

The load balancing of the computational work by Metis provides a very good estimate for the achieved speedup with ParaDyn. However, it is not possible to develop an equivalent estimator for the effect of the communication time on the scalability without running speedup tests with your model on the parallel computer you are using. It is particularly important to do this for models which have those features listed in Section 2.5.

3.0 ANALYSIS WITH PARADYN

An important consideration in developing a parallel simulation capability is the requirement to provide automated tools for partitioning complex finite-element meshes and for managing the hundreds of files generated by a parallel simulation. The ParaDyn software suite includes additional tools for these tasks. This section describes the ParaDyn software products and how to use them.

Parallel simulations have become the best option for producing results for the largest and longest running mechanics simulations and parallel resources are in high demand. Under these circumstances it is very important to benchmark the speedup and scalability for prototypical models and for the parallel computers being used. Once this is done, statistics are generated from the ParaDyn partitioning software to make sure that there is a computational load balance among the processors. Sections 2.3 through 2.6 discuss parallel performance, partitioning issues, speedup, and scalability. Section 2.6 is required reading in order to use the partitioning software effectively and develop efficient parallel models.

3.1 The ParaDyn Software Suite

The ParaDyn program is a message-passing version of the DYNA3D program. The input file for ParaDyn is the same as the DYNA3D input file and can be prepared using the DYNA3D manual and a mesh generator in the usual way. The term *model* used here includes the mesh, all of the initial and boundary conditions, and the selected mechanics options; in other words, all of the data in the input file. The steps for preparing and running a ParaDyn simulation are as follows:

Step 1. Mesh generation and model preparation.
Prepare your input using a mesh generator and the DYNA3D manual.

Step 2. Model Partitioning.
Divide the DYNA3D model into subdomains, one per processor.
Determine the optimal number of processors to use.

Step 3. ParaDyn Analysis.
Run the ParaDyn analysis.

Step 4. Combine the binary output databases for visualization (optional step).

Combine the families of databases from each processor (subdomain) into a family of files in a single database (full domain). This step is used only with the serial visualization tool, GRIZ4S.

Step 5. Visualization.

Display results on the parallel computer or display results on a workstation.

Step 1 is the usual data preparation step for the problem. Step 2 is a preprocessing step for model partitioning and this step is automated with a script file, DynaPart. The DynaPart script runs several programs for dividing the DYNA3D model into subdomains and creating lists of nodes that share results between processors. In Step 3 ParaDyn runs DYNA3D on each of the processors using the subdomain of the mesh assigned to that processor and, when needed, sending nodal results between processors during each timestep of the calculation. Each processor generates a family of text and binary files. Step 4 is used only when post-processing with GRIZ4. The Xmilics utility is used in step 4 to combine the binary database family for each subdomain into a binary database family representing results on the full mesh. In step 5 there are three software products that can be used for visualization: GRIZ4, VisIt, and EnSight. VisIt and EnSight read Mili databases and render images in parallel. EnSight is used at LANL and VisIt is available for use with ParaDyn at LLNL.

The VisIt visualization tool is scheduled for beta testing in early 2004. This tool can be used to visualize the parallel databases *without* the combine step. It can be used on a parallel computer or in a client/server mode with the parallel computer as a server (for parallel rendering and computing) and a PC with a Linux Red Hat operating system or a UNIX workstation as a client (for displaying images, for fast interactivity, and for processing mouse commands). Currently only the Mili format state databases are processed by VisIt. Time history plots generated from the state databases will be available in the next several months.

The software products used for ParaDyn simulations are summarized in Table 1. This table includes the input needed and output generated for each software product.

The next sections describes how to use these products with example execution lines. The final section summarizes the steps for using the ParaDyn software and includes an execute line for each product. This summary can be used as a handy reference once the details of each step are understood.

Task	Software	Input	Output
Partition the model.	dynapart	Dyna3D input file.	Partition map; Plot file for the partitioned mesh.
Run a parallel simulation.	paradyn	Dyna3D input file; Partition map file.	Families of files from each processor include: Plot and time history output; Restart dump files; Text output files.
Combine the binary databases.	xmilics	State and time history databases for each processor (subdomain).	State and time history databases for the full mesh.
Visualize the results.	griz4s visit ensight	State and time history databases for the full mesh. State databases for each processor; Beta testing 01/04. State databases	Screen display; RGB, postscript, PNG and JPEG output files. Client/server screen output, movies, and other graphics formatted output files.

Table 1. ParaDyn Software Products.

3.2 PATH Variable for Accessing ParaDyn Software

The directory containing ParaDyn software must be included in the path names in the UNIX PATH variable. The following **set path** command should be added to the .cshrc file to point to the ParaDyn software products.

```
set path = (/usr/apps/mdg/bin $path)
```

The documentation is located in the directory */usr/gapps/mdg/doc* on all computers at the LLNL computer center. On the engineering servers at LLNL, the documentation is located in the directories, */mdg/manuals* and */public/mdg/doc* for the unclassified and classified networks, respectively.

3.3 Partitioning a Model

The DynaPart script is used for *partitioning* the model input. The model includes the mesh, all of the initial and boundary conditions, and the selected mechanics options; in effect the standard DYNA3D input file. The partitioning subdivides the mesh, contact surfaces, constraints, and other boundary conditions that share nodal data when the domain is divided among the processors. See Section 2 for details.

Running DynaPart in a subdirectory is recommended. This is because the partitioning software generates a number of intermediate files in the current working directory. If these files from the first partitioning of a model are not destroyed, DynaPart with the keyword *again* can be used for subsequent partitioning of the same model for a different number of processors. DynaPart uses results saved from the original DynaPart run and consequently, can skip over some of the time consuming programs executed by the DynaPart script. It is very advisable to use the *again* keyword on the DynaPart execute line when repartitioning a large model. Finally, using a subdirectory also conveniently groups the files containing the results generated for each model that is partitioned.

The form of the DynaPart execute line is

dynapart *infile np [keyword1 keyword2 ...]* **|& tee** *logfilename*

The first two arguments are required. The first argument is the name of the ParaDyn input file and the second argument is the number of processors requested for the partitioning. The remaining arguments are optional keywords. These keywords are described in detail in Appendix 1. The two most commonly used are *again* and *localparallel*. The *again* keyword is described above. The *localparallel* keyword is used in Version 4.1. This keyword will cause large local contact surfaces to be subdivided if a large number of processors is specified. See the discussions in Section 2.4. The output from DynaPart is piped into the UNIX **tee** utility. This step should always be used so that the statistics generated by the partitioning programs as well as other diagnostic messages can be examined after DynaPart is run.

The following is an example of the DynaPart execute lines for the input file *infile* and for partitions for 64 and 128 processors. Notice the second execution line for the same model uses the *again* keyword to save computer time.

dynapart infile 64 |& tee infilelog.64	First partition of a model for 64

	processors
dynapart infile 128 again |& tee infilelog.128	Repartition for 128 processors

The output from the mesh partitioning is a file that is referred to as the *partition* file. The partition file is a required input file for the ParaDyn program.The name of the partition file is *infile.np* where *np* is the number of processors selected. For the *d3samp1* input file, the name for a four-processor partition is *d3samp1.4*. The first several lines in the partition file contain processor (subdomain) totals for the nodes and elements assigned to each processor. This file can be viewed with a text editor.

DynaPart generates a Mili plot database (binary result files) that can be used to visualize the subdomains of the partitioned model. The root name for the plot file is *parplt*. In this special case, there is only one file in the Mili database and this file contains only the geometry for the model. The last character in the geometry file name is always an *A and if you change to a new file name, the last character must also be an A*. GRIZ4 can be used to visualize the database. The result is a coloring of the mesh subdomains by processor and a corresponding map of color values associated with the processors. The color map labeled *materials* is a coloring by processor for plots generated by DynaPart.

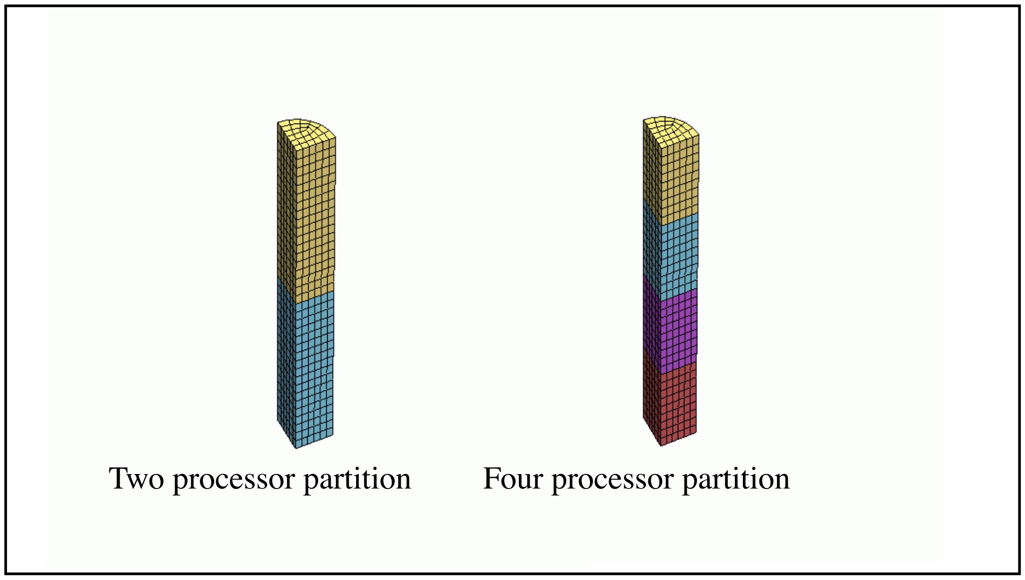


Figure 7. Two and four processor partitions for the *d3samp1* test problem.

Caution: The database file, parpltA, is overwritten when DynaPart is run again in the same directory. The database can be saved by renaming it before partitioning the model another time. Remember the last character in the new name must be an *A*.

There are two important statistics written to the DynaPart log file, 1) the load balance number computed in the Metis programs and 2) if local contact or other special options are used, a processor count is printed which corresponds to the maximum number of processors that can be specified to achieve a computational load balance. The optimal load balance number is 1.000 and generally the load balance should be less than 1.1. See discussions in Section 2.6. These two numbers can be viewed by using the following UNIX **grep** commands on the DynaPart log file as shown in the following examples:

grep "Load balance:" *logfilename*

```
Edge cuts:    1080,  Load balance:  1.00000,  Score: 1.55556.
```

grep "Max # partitions" *logfilename*

```
Max # partitions for good load balance:       27
```

Example 1. Mesh partitioning and repartitioning for a simple model with no contact

This very simple example illustrates the use of DynaPart for model partitioning. This problem does not include the special options, such as contact or nodal constraint sets, that constrain the partitioning of the mesh. Thus, there will not be a statistic computed by DynaPart for the limit on the maximum number of processors.

Step 1. Set up a subdirectory for partitioning the *d3samp1* problem

mkdir partition	Make a subdirectory *partition*
cd partition	
ln -s ../d3samp1 .	Make a link to the input file, *d3samp1*

The UNIX utility **ln** links the *d3samp1* file to the subdirectory *partition* without making a copy. This saves disk space when working with large files. For the same reason it is a good idea to use **ln** in subdirectories when making runs using the same input file and varying the number of processors. This is the case, for example, when generating the speedup curves for a large model.

Step 2. Partition the model for two processors and view the partitioned results with GRIZ4.

```
dynapart d3samp1 2 |& tee d3samp1_log.2
```

The output files from this partitioning are *d3samp1.2*, the plot database, *parpltA*, and the log file, *d3samp1_log.2*. To view the partitioned model, use GRIZ4.

griz4s –i parplt	The root name is used for the input to GRIZ4.
mv parpltA plt2_A	Save the plot file for later viewing. Notice the new plot file name *must* end with *A*.

Step 3. Examine the load balance statistic in the log file

```
grep “Load balance:” d3samp1_log.2
Edge cuts:     360,  Load balance:  1.00000,  Score: 1.18519.
```

Step 4. Repartition the model for four processors.

```
dynapart d3samp1 4 again |& tee d3samp1_log.4
mv parpltA plt4_A
grep “Load balance:” d3samp1_log.4
Edge cuts:    1080,  Load balance:  1.00000,  Score: 1.55556.
```

Step 5. Repartition the model for eight processors.

```
dynapart d3samp1 8 again |& tee d3samp1_log.8
mv parpltA plt8_A
grep “Load balance” d3samp1_log.8
Edge cuts:    2696,  Load balance:  1.02058,  Score: 2.40741.
```

The conclusion to draw from the partitioning for two, four and eight processors is that the optimal number of processors to use for this problem may be four. This can be verified with a scalability study for the particular parallel computer being used.

Example 2. Partitioning a model with contact surfaces

In this example there are ten contact interface definitions. Six of the interfaces are type 2 interfaces and four are type 3 slidelines. In version 2.1 of ParaDyn, each slideline must be fully contained in one processor. In version 4.1 of ParaDyn, the type 3 slidelines can be split into more than one processor if the *localparallel* keyword is used. Future versions of ParaDyn will remove this constraint for type 2 slidelines.

The command lines used to partition and repartition the model for 8, 16, 27, and 32 processors are shown in Table 2. The maximum number of processors for load balancing the calculations is 27. The *load balance statistic* is shown for each of the partitions and shows a significant increase for more than 27 processors.

Table 2. Partitioning and statistics for the df8m3 model

Commands for partitioning the df8m3 model	Comments and statistics
dynapart df8m3 8 |& tee df8m3_log.8 **grep “Load balance:” df8m3_log.8** **grep “Max # partitions” df8m3_log.8**	Eight processor partition. Load balance: 1.03035. The maximum number of processors is 27.
dynapart df8m3 16 again |& tee df8m3_log.16 **grep “Load balance:” df8m3_log.16**	Sixteen processor repartition. Load balance: 1.03075.
dynapart df8m3 27 again |& tee df8m3_log.27 **grep “Load balance:” df8m3_log.27**	Twenty-seven processor repartition. Load balance: 1.03053.
dynapart df8m3 32 again |& tee df8m3_log.32 **grep “Load balance:” df8m3_log.32**	Thirty-two processor repartition. Load balance: 1.18315.

In spite of the good load balance statistics for 27 processors, speedup curves for this problem on several platforms show that it should be run on either 8 or 16 processors!

3.4 File Name Sequences

The ParaDyn program (when restarted) and the post-processing software rely on a knowledge of the naming convention for the families of files generated for each processor in a parallel simulation. The default root names of restart files, plot databases, and text files generated in a ParaDyn run include a processor number in the string. Each processor generates a set of files equivalent to a DYNA3D run on a single processor but with a different root name as shown in Table 3. For a problem running on 1000 or more processors, the length of the string appended to the root name is adjusted to accommodate the number of digits contained in the number of processors used. For example, for 1024 processors, four digits are added to the root name. Similarly for 10240 processors, five digits are added to the root name.

An example showing the file naming conventions including the Mili state and time history databases is shown in Table 3. These are the default names generated for Mili database files for a problem with less than 1000 processors.

Table 3. Database family names generated by ParaDyn

1. State and time history databases: Root names m_p and m_th			
P_0	P_1	P_2	P_3
m_p000A m_p000	m_p001A m_p001	m_p002A m_p002	m_p003A m_p003
m_p00001	m_p00101	m_p00201	m_p00301
m_p00002	m_p00102	m_p00202	m_p00302
...	...	...	...
m_p00099	m_p00199	m_p00299	m_p00399
P_0	P_1	P_2	P_3
m_tht000A m_tht000	m_tht001A m_tht001	m_tht002A m_tht002	m_tht003A m_tht003
m_tht00001	m_tht00101	m_tht00201	m_tht00301
m_tht00002	m_tht00102	m_tht00202	m_tht00302
...	...	...	...
m_tht00099	m_tht00199	m_tht00299	m_tht00399
2. Restart database names: Root names dmp			
P_0	P_1	P_2	P_3
dmp000p01	dmp001p01	dmp002p01	dmp003p01
dmp000p02	dmp001p02	dmp002p02	dmp003p02
dmp000p03	dmp001p03	dmp002p03	dmp003p03
3. Text output file names			

Table 3. Database family names generated by ParaDyn

P_0	P_1	P_2	P_3
d3hsp	d3hsp001	d3hsp002	d3hsp003
frc000	frc001	frc002	frc003

3.5 ParaDyn Execute Line Options

This section discusses the form of the ParaDyn execute line. On most parallel computers the execute line is preceded by a system utility for copying the executable and execute line over to the processors. Specific examples of this are discussed in the sections on running ParaDyn interactively and under a batch processor.

In Version 2.1 the partition file *must* be copied into a file named *partfile* before executing ParaDyn. Do not remove the original partition file because it is used also with the post-processing software. Use a soft link if the partition file is large. In Version 4.1 the partition file can be named either *partfile* or *infile.np* where *np* is the number of processors. This the name generated by DynaPart.

The ParaDyn execute line options are identical to the DYNA3D execute line. There are three options that are important in a parallel simulation: ***q*=**, ***l*=**, and ***c*=**. These options are discussed below and illustrated in the examples. The restarts for ParaDyn likewise use the same execute line as those used in executing a DYNA3D restart.

The execute lines for a ParaDyn initial run and a restart execution are illustrated in the following two examples. The comments relate to the use of the ***q*=**, ***l*=**, and ***c*=** *options* which are described below.

Example 1:

paradyn i=*infile* **l=***filelength*	Specify the file length in Mbytes
paradyn i=*restar*t **r=***dumpfile* **l=***filelength*	Specify the file length on a restart run

Example 2:

paradyn i=*infile* **q=***nseconds*	Allow time for file clean up in a batch run
paradyn c=lastdump q=*nseconds*	Get the restart dump name from file *lastdump*

The first set of execute lines uses the standard input lines for restarts described in the DYNA3D manual. The second set of execute lines illustrates a restart method which starts from the last successfully written restart file.

Notice that it is not necessary to specify an input file for a restart execution, as shown in the second example. If the input file is not specified, all input options will remain the same as those specified in the preceding run. These values from the previous run are stored in the restart dump file and read in during the input phase of the restart run.

The ***l=*** option provides a method for increasing the maximum file length for each member of a database family of plot files. The size specified is in units of Megabytes. *Caution: If the file length option is used on the initial ParaDyn run, it must be used on all subsequent restarts.*

The ***q=*** option provides an important capability for terminating a problem run under batch and providing some extra time for file cleanup. The value, in seconds, specifies the number of seconds (by the wall clock) to run the ParaDyn simulation before ParaDyn stops itself with a normal termination. This option is often used to allow the analyst to specify a wall-clock termination time for a batch run that is shorter than the batch time he selects. The extra time allows for final file writes and close operations for the database families and other output files.

The ***c=*** option is used to get the name of the last *successfully* written restart file. ParaDyn updates a file named *lastdump* with this restart file name after the write is successfully completed. This restart file is then selected by typing **c=lastdump**.

Caution: The standard output files, (d3hsp, d3hsp0001, ...) are overwritten when a subsequent run is made in the same directory. It is generally a good practice to at least save the file *d3hsp* into a different name between restarts of the problem. This output file, *d3hsp*, usually has results that are of value at the end of a long sequence of restarts. For instance, file names, input options, and dynamic relaxation results are written into the file *d3hsp*. Many of the error messages generated by ParaDyn are also written into the *d3hsp* file.

Example: Execute lines for ParaDyn

Consider an initial ParaDyn run using the DYNA3D input file, *d3samp1*and a restart run using an input data file, *irestart*. The initial run and restart run are executed with the following ParaDyn input arguments and files are generated following the naming convention in Table 3.

paradyn i=d3samp1 l=10	File lengths are 10 Mbytes
paradyn i=irestart r=dmp000p01 l=10	Restart includes the **l=** option.

Notice, the file length is specified on both the initial run and on the restart.

This next example illustrates the use of the **q=** termination option.This problem will be submitted to batch to run for one hour (3600 seconds) and ten minutes before the end of the run (after 3000 seconds) ParaDyn will call the termination routine to close files and exit. The maximum length of the files in the plot databases is set to 100 Megabytes. The name of the restart file is the name listed on the first line of the file named *lastdump*.

paradyn i=d3samp1 q=3000 l=100	Allow 10 minutes for file cleanup and 100 Mbyte files
paradyn c=lastdump q=3000 l=100	Restart uses the *lastdump* file,100 Mbyte files and allows 10 minutes for file cleanup

3.6 Running ParaDyn Interactively

The ParaDyn program is usually run under a system utility, such as **mpirun, prun, poe,** or **srun** on a parallel computer. This system utility copies the ParaDyn executable to the processors on a parallel computer. Generally a small number of processors (2 to 16 processors) are available on a *debug* partition for interactive job processing. This gives code developers and analysts an opportunity to develop and test small prototype models prior to running a large model. It is also useful to run scalability studies on a debug partition if it includes enough processors. Once the model is tested, large ParaDyn simulations are submitted for batch processing. Script files are often made available in public file systems to automate batch runs.

3.6.1 Running ParaDyn on an IBM system

ParaDyn can be run interactively using the **poe** utility on the IBM computers at LLNL. In the examples below, the IBM system has four processors per node and five hundred or more nodes. The **poe** utility runs ParaDyn with an input specifying the number of nodes and optionally, the total number of processors. The options for specifying the number of nodes and processors are inserted between the ParaDyn command and the execute line options for ParaDyn. If only one processor per node is being used, then it is sufficient to simply specify the number of nodes with **-nodes** as follows:

poe paradyn -**nodes** *numnodes* -**rmpool 0** *paradyn_execute_line*

More than one processor per node can be used by specifying the number of nodes with -**nodes** and the total number of processors with -**procs**.

poe paradyn -**nodes** *numnodes* -**procs** *np paradyn_execute_line*

Example: Interactive ParaDyn runs using poe

The following are command lines for executing ParaDyn with the *d3samp1* input file and using either four or five processors.

poe paradyn -**nodes 4** -**rmpool 0 i=d3samp1**	Use 1 processor on each of 4 nodes.
poe paradyn -**nodes 1** -**procs 4** -**rmpool 0 i=d3samp1**	Use 4 processors on 1 node.
poe paradyn -**nodes 2** -**procs 5** -**rmpool 0 i=d3samp1**	Use 5 processors on 2 nodes.

3.6.2 Running ParaDyn on an Intel Linux Cluster

Interactive runs on the Linux cluster at LLNL use the utility **srun** for executing ParaDyn.

srun -**N** *numbernodes* -**n** *numberprocessors* -**ppdebug paradyn** *paradyn_execute_line*

Example: Interactive ParaDyn runs using srun

srun -**N 4** -**n 8**-**ppdebug paradyn i=d3samp1**	Use 2 processors on each of 4 nodes.

3.6.3 Running ParaDyn on an SGI Origin

ParaDyn is run interactively using the **mpirun** utility on an SGI Origin computer. The **mpirun** utility runs ParaDyn for *np* processors as follows.

mpirun -**np** *np* **paradyn** *paradyn_execute_line*

Example: Interactive ParaDyn runs using mpirun

mpirun -**np 4 paradyn i=d3samp1**	Use 4 processors.

3.6.4 Running ParaDyn on a Compaq at LANL

The Compaq systems at LANL use the **prun** utility for running interactive problems. Two steps are needed to start up an interactive job. First log into the front end computer and use **bqueues** to find a queue with free processors. Then use the utility **llogin** to login to a parallel partition from the front end computers. Once logged into a partition, the **prun** command starts up the parallel run.

Example: Interactive ParaDyn runs on a Compaq system at LANL

prun -**N 4** -**n 16 paradyn i=d3samp1**	Use 4 processors on each of 4 nodes.

3.7 Running ParaDyn with Batch

ParaDyn production simulations are always run under the batch system. To set up a problem for batch, a script file is prepared and includes the execute lines for ParaDyn with one difference. The the number of nodes and processors is usually specified on a separate line in the script file rather than included as a keyword argument on the execute line. Thus, the execute line needed in the batch script file includes the utility for setting up the parallel execution followed by the standard paradyn execute line. As an example, the script file to run ParaDyn on an IBM includes separate lines for specifying the number of nodes (the -**ln** option in **psub**) and the number of processors (the -**g** option in **psub**). The following are the batch equivalent of the interactive IBM execute lines given in Section 3.6

```
# psub -ln 4
# psub -g 4
poe paradyn i=d3samp1                Use 1 processor on each of 4 nodes.

# psub -ln 1
# psub -g 4
poe paradyn i=d3samp1                          Use 4 processors on 1 node.

# psub -ln 2
# psub -g 5
poe paradyn i=d3samp1                          Use 5 processors on 2 nodes.
```

To set up a problem for batch processing at LLNL, prepare a script file and submit it to the batch system with the **psub** utility. Lines included in a typical script file for a ParaDyn simulation are shown in Figure 8.

The script file is submitted to batch for processing using the PSUB utility.

psub -c *computername scriptfile*

Use the **pstat** utility for interrogating the status of runs submitted with **psub**. A status of RUN indicates the job is running on the parallel computer. The online **man** pages can be viewed to study other options provided by the **psub** and **pstat** utilities. The **spjstat** and **spj** utilities display the status of the job queues and also have **man** pages.

```
# PSUB -eo
# PSUB -tH 2:00          # Job is to run for a maximum of 2 hours
# PSUB -ln 8
# PSUB -g 32
printenv
echo "started at"
date
cd /p/gk2/loginname/dynatests
# Allow 10 minutes (600 seconds) for file cleanup
poe paradyn i=dynin q=6600 l=100
echo "ended at"
date
```

Figure 8. A typical batch script file for a ParaDyn simulation

3.8 Visualizing ParaDyn Results

It is recommended that Mili binary database format be selected when running ParaDyn simulations. The format for the Mili databases provides a considerable amount of flexibility to the code developers for designing material templates that display results specific to each material type as well as the flexibility for selecting (with input keywords) which state data fields should be included in a database generated by a parallel analysis. Recent code additions in ParaDyn and DYNA3D require the use of the Mili database format. In addition, the Mili databases have been extensively tested in large benchmarks and in production runs at both Livermore and Los Alamos. For these reasons, we encourage the use of Mili format binary output. The old Taurus database format is supported only for backward compatibility with old input models.

3.8.1 Selecting Mili Databases with Keyword Input

Mili and Taurus database formats are selected by adding keywords *mili_plot* or *taurus_plot* in the keyword input section of the DYNA3D input file. The keywords for selecting the state databases are:

mili_plot *filename*
endfree
Set the value of the *filename* to the root name desired for the Mili databases. Set the value of *filename* to 1 to select the Mili database format with the default root name, *m_p*. Unless the *taurus_plot* keyword is used, both Mili and Taurus state databases will be generated.

taurus_plot *off*
Disable the output of the Taurus databases.

Example: Specify Mili for the state database format in a ParaDyn analysis.

Add the following keyword input to the keyword control options section of the input file.

mili_plot 1	Write both Mili and Taurus state databases
endfree	

mili_plot 1	Write out Mili state databases
taurus_plot 0	Do not write out Taurus databases

endfree

3.8.2 Combining Parallel Databases and Visualizing Results with GRIZ4

Before the binary databases from a ParaDyn simulation can be visualized with GRIZ4, the processor families must be combined with the utility Xmilics. Then there are two choices for visualizing the results with GRIZ4:

1) The combined databases can be viewed on the parallel computer with GRIZ4.
2) The combined database families can be transported (using FTP or SCP) to a workstation and viewed with GRIZ4 on the workstation.

There are some trade-offs to be made in selecting either of these choices for visualization. For large problems with hundreds of thousands or millions of elements, the rendering and the display of a frame sent over a network to a workstation can be as long as twenty or thirty minutes. In this case better interactivity is possible by using FTP to send the combined database over to a local workstation. The additional resources needed for this improved interactivity are: 1) twice the disk space is needed on the parallel computer, 2) a workstation is needed with high-speed graphics capabilities, and with enough disk space to store the full database from a parallel simulation. Even with this additional hardware, the viewing of results is delayed by the time it takes to FTP the databases to the workstation.

The utility Xmilics combines the Mili state and time history databases and the Taurus state database from a ParaDyn run. The Taurus time history databases must be combined with the older utility COMBINETHS. The combined databases can then be viewed with GRIZ4.

The execute line for Xmilics for both state and time history databases is:

xmilics -i *infileroot* **-o** *outfileroot* **-c** *partfile*

The first argument is the root name for the families of state databases and the second argument is the root name for the output files with the combined databases. The third argument is the name of the partition file.

Example: Xmilics execute line

xmilics -i m_p -o mout -c d3samp1.2	Combine the Mili state database files.
xmilics -i m_tht-o tout -c d3samp1.2	Combine the Mili time history database files.

On the workstation or the parallel computer, these databases are viewed with GRIZ4. Once GRIZ is started up with a Mili database, the GRIZ *load* command can be used to load another Mili database for viewing.

Example: View a State and Time History Database with GRIZ4
View the combined Mili time history and state databases processed with Xmilics in the previous example.

griz4s –i mout	Read and view the state plots.
load tout	Load the time history database for viewing.

Because the Mili time history files are still under development, the Taurus time history files are still supported to provide a backup capability to the time history results available in both the Mili state and time history files. The utility COMBINETHS combines the Taurus time history database from a ParaDyn run. The combined time history database can then be viewed with the THUG utility [11]. The execute line for COMBINETHS is

combineths *infileroot outfileroot partfile* **0 0**

infileroot is the root name for the families of time history databases
outfileroot is the root name for the output files with the combined databases
partfile is the name of the partition file.

The last two arguments must both be 0.

3.8.3 Visualizing Results with VisIt

VisIt is a parallel visualization tool that is currently undergoing beta testing for Mili format databases. VisIt can be run on the parallel computer directly or it can be run in a client/server mode using a workstation or PC as the client and the parallel computer as the server. The VisIt Web page at http://www.llnl.gov/visit includes a *Getting Started* manual as well as a thorough full document. Interactive help information is available with pull down menus in VisIt.

Currently VisIt only supports the Mili state databases. The VisIt plan is to implement time history plots using the Mili state databases in mid or late 2004.

A preprocessing step to VisIt is needed once for each set of Mili databases that will be viewed with VisIt. Use the following VisIt execution to preprocess a Mili database with root name *miliroot:*

visit -makemili *miliroot*	Preprocess Mili database for viewing with VisIt

This will produce a file named *miliroot.mili* which is used by VisIt to open the Mili database family. Once this step is taken, start up VisIt without an argument and open the Mili database by selecting the file *miliroot.mili*.

Example: Visualize the Mili database *m_p* **with VisIt.**

visit -makemili m_p	Create the *m_p.mili* file for VisIt
visit	Start up VisIt for the *m_p* Mili database; Open the file *m_p.mili* in the VisIt file panel.

3.8.4 Visualizing Results with EnSight

EnSight is a commercial parallel visualization tool. The installation of EnSight at LANL includes a Mili file reader. The Xmilics combining step is not needed when using EnSight.

3.9 A Summary of Steps for Running ParaDyn Simulations

Set the UNIX **path** variable to the location for the ParaDyn software in the .cshrc file so that it is included each time a new C shell is started.

set path=(/usr/apps/mdg/bin $path)

The documentation is located in the directory */usr/apps/mdg/doc*.

A summary of steps for executing ParaDyn is shown as a set of steps outlined in tables on the next page.

Steps for Running ParaDyn

Step 1. Partition the mesh and find the optimal number of processors for the run

mkdir partition ; cd partition	Make a directory for partitioning.
ln -s ../d3samp1 .	Link the input file into this directory.
dynapart d3samp1 2 |& tee logfile.2	Run the first partition.
grep "Load Balance:" logfile.2	Check the load balance.
grep "Max # partitions" logfile.2	Check the processor limit.
dynapart d3samp1 4 again |& tee logfile.4	Repartition the model.
mv d3samp1.4 ../	Move the best partition file to the working directory and change back to the work directory.

Step 2. Execute ParaDyn initial and restart runs

mkdir rundir; cd rundir	Make a directory for the simulation.
ln -s ../d3samp1 . **ln -s ../d3samp1.4 partfile**	Link the input and partition files into the directory.
paradyn i=d3samp1 l=100	Initial ParaDyn run with 100 Mbyte file lengths.
paradyn i=irestart d=dmp000p01 l=100	First ParaDyn restart. New input options in file *irestart*.
paradyn i=irestart c=lastdump l=100	Second restart. Use the restart file name in *lastdump* and input options in file *irestart*.
paradyn c=lastdump l=100	Third restart with no resetting of input options.

Step 3. Visualize the results with GRIZ4

xmilics -i m_p -o mout -c partfile	Combine the Mili state databases.
xmilics -i m_th -o tout -c partfile	Combine the Mili time history databases.
ftp myworkstation	Move the combined databases to a workstation.
griz4s –i mout load tout	View the results of the state database. View the results of the time history database with the *load* command.

4.0 INPUT FOR PARALLEL SIMULATIONS

This chapter describes data input and special instructions used for parallel simulations only. Instructions for running multiple ParaDyn analyses and special instructions for post-processing results written to text files are described here.

4.1 Tips for Designing Models with Efficient Parallel Contact

Although ParaDyn can be highly efficient on massively parallel computers, performance of a ParaDyn calculation can be severely degraded by the excessive use of certain slidesurface types and other features. This potential degradation can be a result of load imbalance and/or excess interprocessor communication. This can be mitigated by considering the following tips:

Tip 1. Keep the largest slidesurface small enough to fit in one processor, if possible.
This is especially important for slidesurface types 1, 2, 4, 6, and 7. Nodes and elements associated with a slidesurface of these types are partitioned into one processor, regardless of the size of the slidesurface. Therefore, the minimum computational time per time step cannot be less than the time required for one processor to process these nodes and elements. This lower bound on the computational time per time step limits the maximum speedup possible, and therefore limits the maximum number of processors that can be efficiently used for the calculation.

Tip 2. Keep slidesurfaces of types 3, 5, 8, 9, 10, 12, 13 and 14 as small as possible if subdivided among processors.
A slidesurface of these types is distributed among as many processors as is necessary during partitioning to generate an efficient partition for the model. Distributing the surfaces causes extra interprocessor communication which lowers performance. Small slidesurfaces developed during mesh generation can yield better parallel performance than one large slidesurface that is subdivided by partitioning. Furthermore, the small slidesurfaces can be executed on different processors simultaneously which increases the parallelism.

Tip 3. Avoid slidesurface types 1, 2, 6, and 7.
A slidesurface of any of these types requires that all associated nodes to be contained in one processor. If two slidesurfaces of these types share a node, both slidesurfaces must be computed in the *same* processor. This can lead to an accumulation in one processor of several tied slidesurfaces which have just a few common nodes. This can potentially cause a large load imbalance. If a type

2 slidesurface is used in a model, consider replacing it with a type 9 (tied with failure). If a type 6 slidesurface is used in a model, consider replacing it with a type 8 (nodes spotwelded to surface) slidesurface. If two type 8 or type 9 slidesurfaces share a node, both slidesurfaces can be computed independently in different processors.

Tip 4. Select automatically generated or user-generated surfaces for automatic contact.
By default ParaDyn generates the segments (surface patches) that are used for automatic contact types 12, 13 and 14. However, in some cases the analyst may have defined and optimized the contact surface segments for the model. In this case surface segments can be specified manually for the automatic contact interfaces by using the keyword *segments*. This keyword is added in the keyword input section of the automatic contact interface definition.

segments 1
User-defined slave and master segments are specified. Segments associated with shell elements are treated as shell elements, with thickness, by the automatic contact algorithm.

segments 2 and **segments 3**
User-defined slave and master segments specified. Segments associated with shell elements are treated as brick elements, without thickness, by the automatic contact algorithm, and have an effective depth = (*segments)* x (*thickness)*. This option generates behavior similar to type 3 contact, and can be more robust than the **segments 1** option.

Tip 5. Use domain limiting keywords for segment generation in automatic contact.
Performance may improve if the size of a slidesurface is reduced with domain-limiting keywords. These keywords are added in the keyword input section of the automatic contact interface definition. This list of keywords and values is documented in the DYNA3D User's Manual[1].

xmin,xmax,ymin,ymax,zmin,zmax keywords
Specifiy domain limits for the automatic-contact search.

mat_in mat#1 mat#2 mat#3 … mat#n keyword input
Specify material numbers to include in automatic-contact segment generation.

mat_ex mat#1 mat#2 mat#3 … mat#n keyword input
Specify material numbers to exclude in automatic-contact segment generation.

normal_include mat# px py pz _min _max keyword input
Include only those segments whose angle between the vector from the center of segment to the point (px ,py ,pz) and the segment normal are between *_min* and *_max* degrees.

Tip 6. Develop multiple instances of arbitrary contact and local contact.
For a complex mesh it is beneficial to use both local and arbitrary contact algorithms and to provide multiple instances for each type of contact. For example, defining the full domain for an automatic contact surface can be considerably less efficient than defining several subdomains where the surface motion is contained in the subdomain, although the surface motion is arbitrary. An obvious advantage in doing this is that the regions on a mesh without any contact are not included in either the partitioning for local contact or the search domains for arbitrary contact.

4.2 The NIKE–DYNA Link File

The NIKE-DYNA link file is often used to start up a new run using stress results from a previous run on the same mesh. The details for this technique are described in section 2.19 of the DYNA3D manual. The first run to generate the stress file must include the following free format input to generate the output database with the stresses. The root name for the link file is specified on the following keyword input line.

nikefile *stressfile*
endfree

A ParaDyn execution with this input selecting the name, *str*, for the NIKE-DYNA link file will create database families for each processor labeled

str001, str002, … strppp, …

where *ppp* is the processor number.

A second run uses the same input mesh but other parameters in the input, such as load curves, may be changed. (Although the control options can be changed, it is not a requirement, and the same data input file can be used for both runs.) The second run reads the stress databases, stores the stress data, and uses the deformed nodal coordinate values from the first run. Other parameters associated with the original run may also be set and are described in the section **Keyword-Based Control Features** of the DYNA3D manual.

Example: Generate a NIKE-DYNA link file with ParaDyn

Consider a run with an input file, *d3st*, specifying the name stress on the **nikefile** keyword-input line. A second run will use the generated stress database by specifying the **m=** option on the execute line and new DYNA3D options in the input file, *d3st1*.

paradyn i=d3st
paradyn i=d3st1 m=stress g=glt

In the second run, the plot state database file family is given a new root name, *glt*.

4.3 Multiple Versions of Running Restart Files

The following describes a new capability for generating multiple versions of a running restart dump file. The option is selected with a keyword input and optionally specifying the running restart file names on the ParaDyn execute line. The keyword input is

numrrf *nvers*

The integer value, *nvers*, specifies the number of different versions of the running restart file to save on disk. The default value of *nvers* is 2. The running restart family names are incremented through a set of family members until nvers of them have been written. Once this limit is reached the next running restart over writes the first running restart file in the set. The examples below illustrate the cycling through of the names of the versions of the running restart files.

The number of cycles between the running restart dump files is selected as usual in the fifth data field, columns 36-40, of DYNA3D control card 6. The default names for the running restart file for ParaDyn are *rsfnnnmm*, where *nnn* is the three-digit processor number and *mm* is the family-member number. To select the name of the running restart file, use the **a=** option on the ParaDyn execute line.

Example: Specify multiple versions of the running restart file

Set up the input file to save three different versions of the running restart file.

```
numrrf 3
endfree
```

Suppose a problem is run for 100 cycles and the number of cycles between running restart dumps is 30. The files generated for a 4-processor run will be:

```
rsf00001 rsf00101 rsf00201 rsf00301    (cycles 30)
rsf00002 rsf00102 rsf00202 rsf00302    (cycles 60)
rsf00003 rsf00103 rsf00203 rsf00303    (cycles 90)
```

Suppose this problem is run for another 150 cycles, then the running restart files will be

```
rsf00001 rsf00101 rsf00201 rsf00301    (cycle 120)
rsf00002 rsf00102 rsf00202 rsf00302    (cycle 150)
rsf00003 rsf00103 rsf00203 rsf00303    (cycle 180)
rsf00001 rsf00101 rsf00201 rsf00301    (cycle 210)
rsf00002 rsf00102 rsf00202 rsf00302    (cycle 240)
```

In the above, the cycle (30, 60, 90) files were over written by the cycle (120, 150, 180) files, respectively. And finally, the cycle (210, 240) files over write the cycle (120, 150) files.

Example: Select a new name for the running restart files

Suppose instead in the previous example that the initial problem is restarted from cycle 60 and runs up to cycle 250. A new name can be selected for the versions of the running restart files as follows:

```
paradyn i=inrest,r=rsf00002,a=nrf
```

The files generated will be

```
nrf00001 nrf00101 nrf00201 nrf00301   (cycle 90)
nrf00002 nrf00102 nrf00202 nrf00302   (cycle 120)
nrf00003 nrf00103 nrf00203 nrf00303   (cycle 150)
nrf00001 nrf00101 nrf00201 nrf00301   (cycle 180)
nrf00002 nrf00102 nrf00202 nrf00302   (cycle 210)
nrf00003 nrf00103 nrf00203 nrf00303   (cycle 240)
```

4.4 Nodal Force Output

The nodal force output is described in the keyword input and Section 4.47 of the DYNA3D input manual. Nodal force output is selected using the keyword input variable **nfrout**. The value of this keyword input variable is the total number of nodes that will be written into the output files.

Output format

For each time at which the nodal forces are desired, the following set of data are written:

Card 1 (E12.5,1x,i8)

Col. 1-12	Time at which forces have been computed	E12.5
Col.14-20	Number of nodes (nfrout)	I8

Cards 2 to Card 1+nfrout (i8,3e12.7)

Col. 1-8	Node number	I8
Col. 9-20	Force in the x-direction	E12.5
Col. 21-32	Force in the y-direction	E12.5
Col. 33-44	Force in the z-direction	E12.5

The name of the text output file containing the nodal force data is *nodfrc* for a DYNA3D run.

ParaDyn generates two kinds of text output files. The first output file (generated by processor zero only) contains the times at which the forces are output for the selected nodes. The name of this file is *nfrctimes*. The remaining output data, the node numbers and three components of the force, are written to a set of files generated by all of the processors. The files are named nfrc*nnn*, where *nnn* is the three-digit processor number. By separating the data in this way, the following UNIX utilities can be used to combine the data into a single file, with the time steps for the output at the top, followed by the list of nodes and forces for each time step.

```
cat nfrc* | sort -n - > forceout
```

Example: Combine nodal force output from ParaDyn

The following is an example in which two nodes, (12,144), were selected and written at three different time steps. The nodes were merged using the files from the processors with the previous CAT and SORT utilites. The results are

```
0.00000E+00 2
0.44627E-06 2
0.89344E-06 2
          12  0.00000E+00 0.00000E+00 0.00000E+00
          12  0.00000E+00 0.00000E+00 0.00000E+00
          12  0.00000E+00 0.00000E+00 0.00000E+00
         144 0.00000E+00 0.00000E+00 -0.14000E-11
         144 0.00000E+00 0.00000E+00 -0.14000E-11
         144 0.00000E+00 0.00000E+00 -0.15000E-11
```

To eliminate the time stamp data at the beginning of the file (and read it from the file *nfrctimes* instead in a post-processor), use the following UNIX utilities for combining the output.

```
cat nfrc??? | sort -n  - > forceout
```

5.0 FUTURE ENHANCEMENTS

The following features implemented in DYNA3D are future enhancements in the ParaDyn software.

- Message-passing versions of sliding interface types 1, 2, 6, and 7. These contact surfaces are allocated to one processor by DynaPart in ParaDyn Versions 2.1 and 4.1. Future message- passing versions of the algorithms will provide an option to subdivide these algorithms.

REFERENCES

1. Lin, Jerry I., “DYNA3D: A Nonlinear, Explicit, Dimensional Finite Element Code for Solid and Structural Mechanics—User Manual,” Lawrence Livermore National Laboratory, Livermore, California, UCRL-MA-107254 Rev. 2, **1999**.
2. Speck, D. E., Dovey, D. J., “GRIZ: Finite Element Analysis Results Visualization for Unstructured Grids—User Manual,” Lawrence Livermore National Laboratory, Livermore, California, UCRL-MA-115696, **2000**.
3. Speck, D. E., “MILI: A Mesh I/O Library—Programmer’s Reference”, Lawrence Livermore National Laboratory, Livermore, California, UCRL-MA (in publication) **2000**.
4. Sherwood, Robert J., “DynaPart Auxiliary Documentation Files”, Lawrence Livermore National Laboratory, Livermore, California, UCRL-ID-137888, **2000**.
5. Schauer, D. A., Hoover, C.G., Kay, G. J., Lee, A.S., and De Groot, A.J., "Crashworthiness Simulations with DYNA3D", Paper No. 961249, Transportation Research Board, **1996**.
6. Karypis, G. and Kumar, V., “METIS 3.0: Unstructured Graph Partitioning and Sparse Matrix Ordering System,” University of Minnesota, Department of Computer Science, **1997**. See also http://www-users.cs.umn.edu/~karypis/metis/main.shtml.
7. Karypis, G. and Kumar, V., “A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs,” SIAM Journal on Scientific Computing, **1998**. A short version appears in Intl. Conf. on Parallel Processing, **1995**.
8. Karypis, G. and Kumar, V., “Multilevel k-way Partitioning Scheme for Irregular Graphs,” Journal of Parallel and Distributed Computing, **1997**.
9. Hendrickson, B. and Leland, R., “An Improved Spectral Graph Partitioning Algorithm for Mapping Parallel Computations,” Sandia National Laboratory Report Number SAND92-1460, **1992**.
10. Hoover, C.G., Badders, D. C., De Groot, A.J., and Sherwood, R. J., “Parallel Algorithm Research for Solid Mechanics Applications Using Finite Element Analysis,” Lawrence Livermore National Laboratory, Livermore, California, UCRL-ID-129202, **1997**.

APPENDIX 1. DynaPart Command Line and Keywords

As discussed in Section 3, a DYNA3D input model must be partitioned before using it as input to ParaDyn. The partition file generated by DynaPart includes the lists of nodes and elements assigned to each processor. This file is read by the ParaDyn program and is the map used to distribute the model across the processors on a parallel computer.

1.1 Running the DynaPart software

To access the MDG software, the standard path name must be in the Unix PATH variable. C shell users working on the Livermore Computing systems may add this line to their .cshrc file to set the PATH variable when a new C shell is started.

set path = (/usr/apps/mdg/bin $path)

Online help information that includes the format of the DynaPart command line and descriptions of all of the keywords is available by typing:

dynapart -help

The DynaPart script requires two or more arguments. The first two arguments are mandatory, and the remainder are optional keyword arguments. The first argument is the name of the DYNA3D input file and second argument is the number of processors over which you wish to distribute this job. The syntax of the dynapart command is:

dynapart *<dyna-input-file> <number-processors> [keyword...]*

Items in <> are required. Items in [] are optional.

For example, if the name of the DYNA3D input file is *bigmodel* and you wish to run on 32 processors, cd to the directory containing file bigmodel and issue the following command:

dynapart bigmodel 32

The above command will generate a partition file named bigmodel.32, and will also generate diagnostic output to the user's window. It is useful to save this diagnostic output for future reference. If you wish to save the diagnostic output to file bigmodel32.log, then replace the command above with:

```
dynapart bigmodel 32 |& tee bigmodel32.log
```

Carefully look through the screen output (or the file bigmodel32.log) to make sure that each phase of the partitioning was successful. If error messages were issued, correct the DYNA3D input file and run DynaPart again.

When the partitioning runs to a successful completion, the partition file will be in the current directory, and will have a name of the form <*job-name*>.<*number-of-processors*>. In the example above, the partition file generated is bigmodel.32.

For large models that have long DynaPart run times, it is often possible to repartition the model for a different number of processors using the files generated in a prior run of DynaPart. This can be done if (1) the DYNA3D input file has not changed since the previous partition, (2) the intermediate files from the previous partition are still in the current working directory and have not been modified, and (3) you have not changed the state of keywords *segbased, nosegbased, agglom*, *noagglom*, *localparallel, nolocalparallel, optslide, nooptslide, optslideradius, opslideweight,* or *optslidebuckets* . For a definition of these keywords, see Table 4. The shorter DynaPart run is selected using the keyword *again* on the DynaPart command line. Using this keyword saves time because it eliminates redundant executions of several DynaPart programs:

```
dynapart bigmodel 64 again |& tee bigmodel64.log
```

When using the *again* keyword, the user assumes all responsibility to see that all of the DynaPart intermediate files from the previous partition of this same job remain unmodified since the earlier run.

1.2 Optional Keyword Arguments to DynaPart

There are a number of keyword options that affect the partitioning of an input file. Some keywords modify numerical parameters that are used during partitioning, and other keywords modify the partitioning algorithms in other ways. Recall that the syntax for the dynapart command is:

dynapart *<dyna-input-file> <number-processors> [keyword...]*

Items in <> are required. Items in [] are optional.

There can be any number of keywords, including zero. If no keywords are specified, all keywords assume their default values. Keywords can be specified in any order, except that keywords that specify a value must be followed immediately by that value.

The list of optional keywords in DynaPart is shown in Table 4.

Table 4. DynaPart Keywords

Keyword	Function
again	Rerun DynaPart using intermediate files from the previous run.
segbased	Use segments where appropriate to calculate Special Element sets. Default.
nosegbased	Use nodes rather than segments to calculate Special Element sets.
agglom	Run AGGLOM. Version 2.1 only.
noagglom	Do not run AGGLOM. Default. Version 2.1 only.
localparallel	Subdivide local contact for sliding interface types 3, 5, 8, 9, and 10. Version 4.1 only.
nolocalparallel	Do not subdivide local contact for sliding interface types 3,5, 8, 9, and 10. Version 4.1 only. Default.
optslide	Optimize sliding interfaces by adding edges to the graph. Version 4.1 only. Default.
nooptslide	Do not optimize sliding interfaces by adding edges to the graph. Version 4.1 only.

Table 4. DynaPart Keywords

Keyword	Function
optslideradius <n>	Do not add a graph edge between segments of a sliding interface if they are connected within this distance. Version 4.1 only. Default is 3.
optslideweight <n>	Use edge weight of *n* when adding edges between segments on a sliding interface. Version 4.1 only. Default is 5.
optslidebuckets <n>	Divide the search space for each sliding interface into *n* buckets in each direction to optimize the search for nearest neighbors in the contact search algorithm. Version 4.1 only. Default is 20.
setcostmult <m.n>	Cost multiplier for metavertices. Default is 1.5.
maxsetfillfactor <m.n>	Max size for separable-set segment in units of processor capacity. Range is 0.1 to 1.0 and default is 0.9.
dice	Subdivide separable sets too large for one processor. Default.
nodice	Do not subdivide large separable sets.
metis306	Use Metis V3.0.6 rather than the default Metis.
kmetis	Run kmetis instead of default skmetis.
pmetis	Run pmetis instead of default skmetis.
skmetis	Run skmetis. Default.
multiconstraint	Run multiconstraint Metis instead of default single constraint.
partitioncommcost <m.n>	Communication cost for edge cut. Default 0.5.
partitiontimes <n>	Number of times to partition for each set of SKMETIS parameters. Default 4.
partitionverbosity <n>	Amount of status information from SKMETIS: 1 = minimum 2 = each parameter set 3 = each iteration. Default is 1.
plotnone	Do not generate a plot file of the partition.
plottaurus	Generate a Taurus plot file of the partition.

Table 4. DynaPart Keywords

Keyword	Function
plotmili	Generate a Mili plot file of the partition. This is the default.
plotboth	Generate Taurus and Mili plot files of the partition.

The following is a discussion of each of the keywords.

again

This keyword eliminates redundant initial steps when repartitioning a data set that has been partitioned earlier. It may *not* be used if changing the status of keywords *segbased*, *nosegbased*, *agglom*, *noagglom*, *localparallel, nolocalparallel, optslide, nooptslide, optslideradius, optslideweight* or *optslidebuckets*, but may be used if changing the number of partitions or the values of other keywords. If *again* is not specified, the default is to repeat all steps of the partitioning.

segbased

Use segments (also known as facets) to generate sets of elements that must be handled in a single processor. Elements are included only if they contain all four nodes of a segment in a special DYNA3D object such as a sliding interface. This is the default.

nosegbased

Use nodes to generate sets of elements that must be handled in a single processor. Elements are included if they contain any node in a special DYNA3D object such as a sliding interface.

agglom

Keyword for ParaDyn Version 2.1 only. Run program AGGLOM to generate a graph that better represents models having automatic contact sliding interfaces.

noagglom

Keyword for ParaDyn Version 2.1 only. Do not run program AGGLOM.

localparallel

Keyword for ParaDyn Version 4.1 only. Subdivide local contact surfaces for sliding interface types 3, 5, 8, 9, and 10. This keyword provides the option to use more processors for models that previously would not partition well because the largest local contact surface(s) were constrained to reside fully in one processor. This is the default.

nolocalparallel
Keyword for ParaDyn Version 4.1 only. Do not subdivide local contact surfaces for sliding interface types 3, 5, 8, 9, and 10.

optslide
Keyword for ParaDyn Version 4.1 only. Program LINGRF will optimize sliding interfaces by adding edges to the graph. These edges connect nearest neighbor segments in the sliding interface which are assumed to be on opposite sides of the sliding interface. This is the default.

nooptslide
Keyword for ParaDyn Version 4.1 only. Do not optimize sliding interfaces by adding edges to the graph.

optslideradius <n>
Keyword for ParaDyn Version 4.1 only. When adding edges between segments on a sliding interface, do not add an edge if the proposed segment is connected to this segment within a distance of *n* segments. Default is 3.

optslideweight <n>
Keyword for ParaDyn Version 4.1 only. When adding edges between segments on a sliding interface, set the edge weight to *n*. Default is 5.

optslidebuckets <n>
Keyword for ParaDyn Version 4.1 only. To optimize the search for nearest neighbors in sliding interfaces, we subdivide the search space into *n* buckets in each direction. Default is 20.

setcostmult <m.n>
Specify a floating point cost multiplier for special sets. This multiplier is a floating point number of value greater than 1.0 to represent the extra computational burden imposed by DYNA3D objects generating special sets (such as sliding interfaces). Default is 1.5.

maxsetfillfactor <m.n>
Maximum size for a segment of a separable set, in units of processor capacity. If *dice* is specified, program CUTSEP will cut large separable sets into segments that are no larger than this. The value for maxsetfillfactor should be between 0.1 and 1.0, and defaults to 0.9. This is ignored if keyword *nodice* is used.

dice
Program CUTSEP will subdivide (dice) separable sets if they are too large for one processor. Separable sets are generated by automatic contact sliding interfaces and also by local contact sliding interfaces if the *localparallel* keyword is specified. If a separable set is too large to fit in one processor, DynaPart will determine how many segments it needs to be cut into, and will cut it optimally. This is the default.

nodice
Do not subdivide large separable sets, even if they are too large for one processor. If such large separable sets are present, the resulting partition will have a suboptimal load balance.

metis306
At the heart of the partitioning software is a graph partitioning program called Metis, from the University of Minnesota. The latest release of Metis is Version 4 or higher, but some users may prefer partitions that are performed by Metis Version 3.0.6. Specifying keyword *metis306* will cause DynaPart to run Version 3.0.6 of pmetis or kmetis. If you do *not* specify *metis306*, DynaPart will default to using the most recent version of Metis on the system.

The three metis keywords specifying a Metis program (*kmetis*, *pmetis* or *skmetis*) are mutually exclusive. Only one may be specified. If none are specified, *skmetis* is used.

kmetis
Use kmetis rather than skmetis or pmetis. kmetis uses the k-way partitioning algorithm.

pmetis
Use pmetis rather than skmetis or kmetis. pmetis uses the recursive partitioning algorithm. This is the default if metis306 is specified.

skmetis

Use skmetis rather than kmetis or pmetis. skmetis runs several variations of the kmetis routine, then selects the result having the best load balance. This option takes more time and memory to partition than *pmetis* or *kmetis* do, but may result in a better partition. This is only available in Metis Version 4 and higher. This is the default unless metis306 is specified.

multiconstraint
Run multi-constraint Metis. Use at your own risk. This will result in more time spent running Metis, and is only available if *metis306* is *not* specified. Multi-constraint Metis may produce a better partition. The default is to run single-constraint Metis.

partitioncommcost <m.n>
skmetis computes the quality of a partition based on both load balance and the number of graph edge cuts (which correspond to interprocessor communications). Keyword *partitioncommcost* is used to specify a real number that is the multiplier of the number of edge cuts when computing the partition score. It is normally between 0.0 and 1.0, but may be larger. Value 0.0 means edge cuts are not considered in the score. Default is 0.5.

partitiontimes <n>
skmetis varies several parameters and computes partition(s) for each set of parameter values. For a given set of parameters, partitions are computed *partitiontimes* times, with a different state of the random number generator for each calculation. After all partitions with all values of parameters have been computed, the partition with the best score is returned. Default is 4.

partitionverbosity <n>
The amount of reporting from skmetis can have one of three values:

partitionverbosity	1 = Minimum reporting. Report the statistics and the control parameters that generated the best partition.
partitionverbosity	2 = Medium reporting. In addition to (1), for each value of control parameters, report summary statistics (ranges of load balances and edge cuts) for the partitions that were computed.
partitionverbosity = 3	3 = Full reporting. In addition to (2), report the statistics (load balance and edge cuts) for *each* partition that was calculated. Default is 1.

plotnone

Do not generate a plot file of the partition. If *plotnone* is specified, no other plot options may be specified.

plottaurus
Generate a Taurus format plot file of the partition. Some users may prefer this legacy file format.

plotmili
Generate a Mili format plot file of the partition. This is the preferred format, and is the default if no plot keywords are supplied.

plotboth
Generate both Taurus and Mili format plot files of the partition. If *plotboth* is specified, no other plot options may be specified. Specifying *plotboth* is equivalent to specifying both *plottaurus* and *plotmili*.

Table 5. DynaPart Keyword Defaults and Overrides

Parameter	Default	Override with keyword
Faster repartition?	Rerun all	*again*
Segment-based contact?	Yes	*nosegbased*
Run AGGLOM? Version 2.1 only.	No	*agglom*
Subdivide local parallel contact types 3, 5, 8, 9, and 10? Version 4.1 only.	Yes	*nolocalparallel*
Add graph edges across sliding interfaces? Version 4.1 only.	Yes	*nooptslide*
Minimum segment count for adding a graph edge. Version 4.1 only.	3	*optslideradius <n>*
Weight of each added graph edge. Version 4.1 only.	5	*optslideweight <n>*
Number of search buckets in each direction for finding graph edges to add. Version 4.1 only.	20	*optslidebuckets <n>*
Set-cost multiplier for meta-vertices.	1.5	*setcostmult <m.n>*

Table 5. DynaPart Keyword Defaults and Overrides

Parameter	Default	Override with keyword
Maximum separable-set chunk	0.9	*maxsetfillfactor <m.n>*
Subdivide large separable sets?	Yes	*nodice*
Metis version	Latest	*metis306*
Metis program	skmetis	*kmetis, pmetis*
Multiconstraint Metis?	Single	*muliconstraint*
skmetis communication cost	0.5	*partitioncommcost <m.n>*
skmetis # times each parameter set	4	*partitiontimes <n>*
skmetis verbosity	1 (low)	*partitionverbosity <n>* 2 (medium), 3(full)
Plot file format	Mili	*plotnone*, *plotboth*

1.3 Using the *dice/nodice* and *agglom/noagglom* Keywords

Keywords *agglom/noagglom* are available in ParaDyn Version 2.1 only. Version 4.1 of ParaDyn eliminates the *agglom/noagglom* keywords and substitutes a set of keywords for optimizing the subdividing of selected sliding interfaces. See the set of *optslide* keywords in Table 4.

Keywords *dice/nodice* are available in both ParaDyn Versions 2.1 and 4.1.

The following are guidelines to help the analyst decide whether to use the *dice* or *nodice* keywords and the *agglom* or *noagglom* keywords.

- If not specified, the software presently takes the defaults of *dice* and *noagglom*. This is subject to change in future releases.
- The *agglom* keyword causes program AGGLOM to be run. AGGLOM is an experimental program that computes the elements involved in automatic contact when segments are *not* supplied in the input file. It also enhances the graph representing the model in a manner that helps assure a partition with lower communication (a faster partition).
- *Separable sets* are generated by sliding interfaces of Type 12, 13 and 14 (automatic contact sliding interfaces). In ParaDyn Version 4.1, sliding interface types 3, 5, 8, 9, and 10 are also defined as separable sets if keyword *localparallel* is used.

- If there are no large separable sets present, simply default (do not specify) either of the keywords. There is nothing to be gained by running AGGLOM (hence we'll accept the default *noagglom*), and there is no harm in using *dice* (the default). The software run by the *dice* option will have no effect in this situation.
- If there are one or more large separable sets present that correspond to automatic contact sliding interfaces with segments *not* defined (*segments* zero or not defined in the input file), then either:
  a). Specify neither keyword (thus in effect supplying *dice* and *noagglom*). In this case, such sliding interfaces will potentially be spread across all processors, and will run correctly but with a high communication overhead (a slower-running partition), or
  b). Specify keyword *agglom* and use *dice* (the default). This will run program AGGLOM to compute the elements associated with the automatic contact, then will spread those elements across as few processors as possible. This should result in a higher-performance run.
- If there are one or more large separable sets present that correspond to automatic contact sliding interfaces with segments defined (*segments* nonzero in the input file), then:
  a). *dice* should be used to subdivide the large separable sets, (however *dice* is the default, so does not need to be specified).
  b). If *agglom* is not specified, such separable sets will partition satisfactorily.
  c). If *agglom* is specified, the resulting partition may have less communication, hence run more rapidly.

1.4 Example DynaPart execution lines

1. Partition input file *bigmodel* for 32 processors, using all the default parameters:

```
dynapart bigmodel 32
```

2. Partition *bigmodel* for 32 processors, and run AGGLOM:

```
dynapart bigmodel 32 agglom
```

3. Partition *bigmodel* for 32 processors, but do not subdivide large separable sets (such as automatic contact sliding interfaces), even if they are too large for one processor:

```
dynapart bigmodel 32 nodice
```

4. Partition *bigmodel* for 32 processors. If separable sets are too large for one processor, subdivide them as necessary:

dynapart bigmodel 32 dice
or
dynapart bigmodel 32

5. We discovered from the results of partitioning Example 4 that even though the largest special element set was from a separable set, the load balance still was higher than desired. Now we want to repartition, making the largest such set only 0.8 of a processor's worth:

dynapart bigmodel 32 dice maxsetfillfactor 0.8 again

6. Partition *bigmodel* for 32 processors, using kmetis Version 3.0.6:

dynapart bigmodel 32 metis306 kmetis

7. Partition *bigmodel* for 32 processors, using pmetis:

dynapart bigmodel 32 pmetis

8. Partition *bigmodel* for 32 processors, using multiconstraint skmetis:

dynapart bigmodel 32 multiconstraint

9. Partition *bigmodel* for 32 processors, but multiply the estimated compute load to compute sliding interfaces and other special element sets by a factor of 1.83:

dynapart bigmodel 32 setcostmult 1.83

10. Partition *bigmodel* for 32 processors, and generate a Taurus format plot file:

dynapart bigmodel 32 plottaurus

11. Partition *bigmodel* for 32 processors, and generate both Taurus and Mili format plot files:

dynapart bigmodel 32 plottaurus plotmili

or

dynapart bigmodel 32 plotboth

12. Partition *bigmodel* for 32 processors, without using segments:

dynapart bigmodel 32 nosegbased

APPENDIX 2. DynaPart Log File

This appendix provides more detailed description of the background processing in DynaPart. It is intended to expand on the examples provided in Section 3.0.

Up-to-date information about the latest changes in DynaPart is available online in the standard documentation directory. The files related to DynaPart in this directory are:

- HOW-TO-PARTITION: This provides instructions for running DynaPart, including a complete list of keywords, their meanings, default values, and examples.
- FOR-GOOD-PARTITIONS: This lists the DYNA3D objects which constrain partitioning, and how to interpret DynaPart log files.
- PARTITION-FILE-FORMAT: This file documents changes to the partition file format.

2.1 DynaPart Log File

A complete log file from a DynaPart execution is shown below. The two statistics of interest to the analysts running the problem are highlighted with underlined, bold text. These statistics are 1) the maximum number of processors (partitions) for good computational load balance for contact and other special options printed by program reducegrf and 2) the compuational load balance printed by program skmetis. The programs run by the DynaPart script depend on the mesh and boundary conditions for the specific problem being partitioned, as explained in APPENDIX 3. The programs executed in DynaPart for this particular problem are highlighted with double-underlined text.

The first part of the log file includes the following items:

- The DynaPart options selected either by default or from keywords specified on the execution line. See Appendix 1 for details about keywords.
- The file extensions for files generated during a partitioning. The files generated by DynaPart use the input file name as a root name and use an extension for each type of file generated during the partitioning process.
- Statistics and other output generated by the programs run in the DynaPart script.

This is followed by the output from each of the programs executed by DynaPart. DynaPart was run was run with the following execution line and generated a log file *df8m3.log.8*:

dynapart df8m3 8 | tee df8m3.log.8

DynaPart Log File for problem *df8m3*

```
Partitioning df8m3 for 8 processors.

DynaPart options:
  Use segments where appropriate to generate Special Element file.
  Do NOT run AGGLOM.
  Subdivide large separable sets (e.g. autocontact sliding interfaces).
    Make each piece no larger than 0.9 of a processor's worth.
  Use current default version of skmetis.
  Use single-constraint skmetis.
  Cost of an edge cut by skmetis is 0.5.
  Run skmetis 4 times for each parameter set.
  Multiply cost of sets (e.g. sliding interfaces) by 1.5.
  Write a Mili format plot file.

This script executes the software in the following order:
    SNPGEN - Generates sets of special nodal points, segments,
             set types, and free format keyword sections.
    SELSETS - Selects the ganged sets from the SNPGEN output
    DUALGEN - Generates a dual graph file, special elements (SE) file
              enhanced segment file, vertex weight file and AGGLOM files.
    SETCOST - Computes cost of each SE set
    AGGLOM - Adds edges to the graph and elements to the SE file
    CUTSEP - Cuts oversized separable sets
    EXTGRF - Extracts subgraphs from a graph
    CTS - Converts a color file and its new-to-old map to a SE file
    TRANSCLOSE - Transitively closes the ganged special elements
    AETS - Assigns special elements to disjoint sets
    REDUCEGRF - Reduces the graph by reducing SE sets to a meta-element
    SKMETIS - Partitions the reduced graph, produces a color file
    EXPANDCOL - Expands the color file to include all original elements
    SETCOL - Specifies color for all special element sets
    SELSETS - Selects the ganged and unganged sets from the SNPGEN output
    SELCOLS - Selects colors for ganged and unganged sets from SNPGEN
    PFGEN - Generates a partition file

Internal and output file extensions:
  .snp             Special Nodal Points (SNP) file (from snpgen)
  .seg             Segment file (from snpgen)
  .sst             Special Sets Type file (from snpgen)
  .ssff            Special Sets Free-Format file (from snpgen)
  .nsnp            Number of Special Nodal Points sets (from snpgen)
  .gsnp            Ganged Special Nodal Points file (from selsets)
  .grforig         Dual graph file (from dualgen)
  .se              Special Element (SE) file (from dualgen)
  .eseg             Enhanced segment file (from dualgen)
  .vwt              Vertex weight file (from dualgen)
  .edb              Element database file (from dualgen)
  .ndb              Node database file (from dualgen)
  .secost           Special element set cost file (from setcost)
  .grfee            Graph file with additional edges (from agglom)
  .seee             SE set with extra elements file (from agglom)
  .sew              Special element set weight file (from agglom)
  .seeecost         SE set with extra elements cost file (from setcost)
  .bigsep           Oversized separable SE sets (from cutsep)
  .grfsep<M>        Subgraph of vertices from big SE set <M> (from extgrf)
  .grfsep<M>.part.<N> Local color file (from metis) for big SE set (metis)
  .secut<M>         SE file of big SE set <M> after cutting (from cts)
  .sesep            SE file after big separables cut (from cutsep)
  .sstsep           Special Set Types file after cutting (from cutsep)
  .nosesep          New-Old Special Element set mapping (from cutsep)
  .setcg            Special Elements Transitively Closed Gang file
                      (from transclose)
  .ontcgs           Old-New TransClosed Gang Set mapping file
                      (from transclose)
  .dse              Disjoint Special Elements file (from aets)
  .onses            Old-New Special Element Set mapping file (from aets)
  .grfred           Reduced Graph file (from reducegrf)
  .one              Old-New Element mapping file (from reducegrf)
  .grfred.part.<N> Local color file (from metis)
  .part.<N>         Global color file (from expandcol)
  .snscol           Special Nodal Set Color file (from setcol)
  .uggsnp           SNP file from unganged and ganged only (from selsets)
  .uggsnscol        Color file for SNP sets from unganged/ganged only
                      (from selcols)
  .<N>              Partition file (from pfgen)
  parplt            Taurus-format partition plot file (from pfgen)
  parpltA           Mili-format partition plot file (from pfgen)

Thu Nov  6 12:09:00 PST 2003
```

Running SNPGEN

```
        ****  SNPGEN - Generate a Special Nodal Points file  ****
                         Version of  9-Apr-2002

  DIPOLE FACADE 8 (ppp wes-ssa)

 <<  Starting to read the Dyna3d input file.

          Number of materials:                              7
          Number of nodal points:                      157331
          Number of 8-node hexagonal elements:         142369
          Number of 2-node beam elements:                9646
          Number of 4-node shell elements:                  0
          Number of 8-node thick-shell elements:            0

 >>  Finished reading the Dyna3d input file.
 --  Starting to sort the special nodal points lists.

 ++  Finished sorting the special nodal points lists.

 --  Starting to write the special nodal points output file.

 ++  Finished writing the special nodal points output file.

 --  Starting to write the segment output file.

 ++  Finished writing the segment output file.

 --  Starting to write the set type output file.

 ++  Finished writing the set type output file.

   There were:
         10 Special Nodal Point sets.  The longest set contained
       2208 nodal points, and all sets combined contained a total of
      12690 nodal points.
   There were:
         10 segment sets.  The longest set contained
       2046 segments, and all sets combined contained a total of
      13938 segments.
1.32u 0.12s 0:08 17% 0+7k 3329+98io 146pf+0w
```

Running SELSETS

```
       ****  SELSETS - Select sets of the specified type(s)  ****
                        Version of  1-Jul-2002

 Selecting sets of type 2.

 << Starting to read the sets input file

 >> Finished reading the sets input file

 << Starting to read the set types input file

 >> Finished reading the set types input file

 -- Starting to write the selected sets output file

 ++ Finished writing the selected sets output file

 There were:
          6 selected sets.  The longest set contained
       1998 elements, and all sets combined contained a total of
       5661 elements.
0.02u 0.00s 0:00 13% 0+3k 10+8io 2pf+0w
```

Running DUALGEN

```
           ****  DUALGEN - Generate a dual-mesh graph file  ****
```

```
                        Version of 11-Jul-2002

  DIPOLE FACADE 8 (ppp wes-ssa)

 <<  Starting to read the Dyna3d input file.

          Number of materials:                           7
          Number of nodal points:                   157331
          Number of 8-node hexagonal elements:      142369
          Number of 2-node beam elements:             9646
          Number of 4-node shell elements:               0
          Number of 8-node thick-shell elements:         0

 >>  Finished reading the Dyna3d input file.

 --  Starting to generate the node-to-element connectivity array.

 ++  Finished generating the node-to-element connectivity array.

 --  Starting to generate Special Element and enhanced segment files.

     These Special Element sets DO share some elements.
     You should run TRANSCLOSE to combine SE sets having shared elements.

 ++  Finished generating Special Element and enhanced segment files.

 --  Starting to generate the element-to-element connectivity array.

 ++  Finished generating the element-to-element connectivity array.

 --  Starting to generate the dual-mesh graph file.

     Min elem cost at elem         1 orig cost    1.00 scaled cost       1.
     Max elem cost at elem         1 orig cost    1.00 scaled cost       1.

     Total cost of the unscaled graph was           152015.
     Total cost of the scaled graph is              152015.

 ++  Finished generating the dual-mesh graph file.

 --  Starting to generate the vertex weight file.

 ++  Finished generating the vertex weight file.

 --  Starting to generate the vertex geometry file.

 ++  Finished generating the vertex geometry file.

     Maximum element connectivity of   56 seen at element  131795.

 ..  Writing element data to database file -> df8m3.edb

 ..  Writing node data to database file -> df8m3.ndb
16.83u 3.83s 0:27 75% 0+148k 5053+6831io 17pf+0w
```

Running SETCOST

```
              ****  SETCOST - Compute costs of sets  ****
                         Version of 20-Feb-2002

 << Starting to read the special element set input file

 >> Finished reading the special element set input file

 << Starting to read the element weight input file

 >> Finished reading the element weight input file

 -- Starting to write the set costs output file

 ++ Finished writing the set cost output file

 Set cost statistics:
   Number of sets:                                         10
   Cost of most costly set:                              4248
   Total cost of all sets:                              17608
   Total cost of all vertices in graph:                152015
0.38u 0.00s 0:00 90% 0+8k 9+2io 1pf+0w
```

Running CUTSEP

```
    CUTSEP will subdivide large separable sets by an additional factor of 1.

    ****  CUTSEP - Find and subdivide the oversize separable sets  ****
                          Version of 20-Feb-2002

 << Starting to read the special element set input file

 >> Finished reading the special element set input file

 << Starting to read the sets cost input file

 >> Finished reading the sets cost input file

 << Starting to read the sets type input file

 >> Finished reading the sets type input file

    Cost of entire graph is 152015.
    Number of special elements before being made disjoint is 17608.
    Estimated (probably high) cost of reduced graph is 160819.
    Number of processors is 8.
    Set cost multiplier is  1.50.
    Maximum set fill factor is  0.90.
    Therefore, cost of largest permissible separable set is 12061.

 -- Starting to write the new-to-old set mapping output file
```

```
    There were no oversized separable sets.
    Input special element (SE) file has been copied to output SE file.
    Input special sets type (SST) file has been copied to output SST file.
    The new-to-old set mapping is the identity mapping.

 ++ Finished writing the new-to-old set mapping output file

0.03u 0.02s 0:00 29% 0+2k 15+28io 3pf+0w
```

Running TRANSCLOSE

```
      ****  TRANSCLOSE- Perform transitive closure on ganged sets  ****
                         Version of 17-Jan-2001

 <<  Starting to read the sets input file

 >>  Finished reading the sets input file

     There were a total of   17608 values read, in      10 sets.

 <<  Starting to read the set type input file

 >>  Finished reading the set type input file

 --  Starting to calculate transitively closed ganged sets

 ++  Finished calculating transitively closed ganged sets

 --  Starting to write the set trans-closed gang output file

 ++  Finished writing the set trans-closed gang output file

 --  Starting to write the old-to-new set mapping output file

 ++  Finished writing the old-to-new set mapping output file

 There were:
        10 output sets.  The longest set contained
      8882 elements, and all sets combined contained a total of
     16508 elements.

0.04u 0.01s 0:00 22% 0+12k 39+28io 4pf+0w
```

Running AETS

```
                  ****  AETS - Assign elements to sets  ****
                            Version of 12-Feb-2002

 << Starting to read the transitively closed gang set input file

 >> Finished reading the transitively closed gang set input file

 << Starting to read the sets type input file
 >> Finished reading the sets type input file

 << Starting to read the old-to-new set mapping input file

 >> Finished reading the old-to-new set mapping input file

 -- Starting to write the disjoint sets output file

 ++ Finished writing the disjoint sets output file

 -- Starting to write the old-to-new set mapping output file

 ++ Finished writing the old-to-new set mapping output file

 There were:
         4 disjoint sets.  The longest set contained
      3941 elements, and all sets combined contained a total of
     15764 elements.
0.04u 0.00s 0:00 33% 0+7k 15+21io 3pf+0w
```

Running REDUCEGRF

```
                 ****  REDUCEGRF - Reduce a graph file  ****
                           Version of 24-Jun-2002

 << Starting to read the special element input file

 >> Finished reading the special element input file

 << Starting to read the graph input file

 >> Finished reading the graph input file

    There were 43851 connections between meta-vertices.

 -- Starting to sort each row of the output graph

 ++ Finished sorting each row of the output graph

 -- Starting to write the graph output file

 Output graph statistics:
  Sum of vertex weights:                      159899
  Largest vertex weight:                        5912
    at reduced vertex:                        136252
    which represents special element set:          1
  Max # partitions for good load balance:         27

 ++ Finished writing the graph output file

18.43u 0.33s 0:20 91% 0+286k 14+3723io 3pf+0w
```

Running SKMETIS

```
Completed 1 / 4 of the iterations.
Completed 2 / 4 of the iterations.
Completed 3 / 4 of the iterations.
Completed 4 / 4 of the iterations.

**************************************************************************
  SKMETIS 4.0.1 Copyright 2001, 2002 The Regents of the
    University of California.  All rights reserved.  Based on
  METIS 4.0.1 Copyright 1998, Regents of the University of Minnesota

Graph Information -----------------------------------------------------
  Name: df8m3.grfred, #Vertices: 136255, #Edges: 1646889, #Parts: 8.
  Total vertex weight: 159899.

Scoring and Calculation Information -----------------------------------
  Edge cut factor: 0.500,  Number of iterations per parameter set: 4,
  Total number of iterations: 48.

K-way Partitioning... ----------------------------------------------
  This best iteration is using options [0 to 4]: 1 4 1 1 0,  offset: 0.
  Edge cuts:  279326,  Load balance:  1.03035,  Score: 1.90380.

Timing Information (Seconds) -----------------------------------------
  I/O:                   1.817
  Partitioning:         35.432   (SKMETIS time)
  Total:                37.365
  Note: If the time was greater than 35 minutes, ignore the above times.
**********************************************************************
37.08u 0.29s 0:39 95% 0+783k 38+34io 10pf+0w
```

Running EXPANDCOL

```
                 ****  EXPANDCOL - Expand a color file  ****
                           Version of 15-Jan-1999

 << Starting to read the global-local elements input file

 >> Finished reading the global-local elements input file

 << Starting to read the reduced color input file

 >> Finished reading the reduced color input file

 -- Starting to write the color output file

 ++ Finished writing the color output file

0.72u 0.00s 0:00 87% 0+10k 8+39io 2pf+0w
```

Running SETCOL

```
    ****  SETCOL - Generate a color file for the special sets  ****
                         Version of  1-Jul-2002

 << Starting to read the old-to-new set mapping input file
 >> Finished reading the old-to-new set mapping input file

 << Starting to read the reduced graph color input file

 >> Finished reading the reduced graph color input file

 -- Starting to write the set color output file

 ++ Finished writing the set color output file

0.13u 0.00s 0:00 72% 0+2k 7+1io 2pf+0w
```

Running SELSETS

```
      ****  SELSETS - Select sets of the specified type(s)  ****
                       Version of  1-Jul-2002

 Selecting sets of type 1 or 2.

 << Starting to read the sets input file

 >> Finished reading the sets input file

 << Starting to read the set types input file

 >> Finished reading the set types input file

 -- Starting to write the selected sets output file

 ++ Finished writing the selected sets output file

 There were:
        10 selected sets.  The longest set contained
      2208 elements, and all sets combined contained a total of
     12690 elements.
0.03u 0.00s 0:00 60% 0+2k 1+16io 0pf+0w
```

Running SELCOLS

```
 ****  SELCOLS - Select colors for sets of the specified type(s)  ****
                       Version of  8-Jul-2002

 Selecting sets of type 1 or 2.

 << Starting to read the set types input file

 >> Finished reading the set types input file

 << Starting to read the set colors input file

 >> Finished reading the set colors input file

 -- Starting to write the set colors output file
 ++ Finished writing the set colors output file

 There were 10 selected colors.
0.00u 0.00s 0:00 0% 0+0k 8+1io 0pf+0w
```

Running PFGEN

```
          ****  PFGEN - Generate a partition-assignment file  ****
                         Version of  9-Apr-2002

  DIPOLE FACADE 8 (ppp wes-ssa)

 <<  Starting to read the Dyna3d input file.

          Number of materials:                                 7
          Number of nodal points:                         157331
          Number of 8-node hexagonal elements:            142369
          Number of 2-node beam elements:                   9646
          Number of 4-node shell elements:                     0
          Number of 8-node thick-shell elements:               0

 >>  Finished reading the Dyna3d input file.

 <<  Starting to read the partition-color file.

          Number of processors in the color file:              8

 >>  Finished reading the partition-color file.

 <<  Starting to read the gang node set file

 >>  Finished reading the gang node set file

 <<  Starting to read the pre-assigned node set file.

 >>  Finished reading the pre-assigned node set file.

 --  Starting to generate the partition-assignment file.

     There are no unassigned nodes to assign.

                                 Number of      Number of
                                   Node         Adjacent
                               Communications   Processors

        Average per processor:      7844.0           4.2
         Max on any processor:     18277            7
      Total on all processors:     62752           34


 ++  Finished generating the partition-assignment file.

 --  Starting to generate the Mili plot file.
 ++  Finished generating the Mili plot file.

5.06u 0.37s 0:06 77% 0+45k 580+1364io 14pf+0w

Thu Nov  6 12:10:44 PST 2003
Partition file name is df8m3.8.
All done
```

APPENDIX 3. Generating Good Partitions with DynaPart

Decisions made during the development of a DYNA3D/ParaDyn model can affect the scalability and efficiency of the simulation. The following are some insights to give the analyst an idea of the scalability of a ParaDyn job, and some tips on how to improve that scalability.

3.1 Special Node and Element Sets

The partitioning of a DYNA3D/ParaDyn model is constrained by boundary conditions and other options contained in that model. These boundary condtions and options will force nodes and elements to be assigned to a single processor rather than being divided across more than one processor. Nodes that need to be kept together are assigned to Special Nodal Point (SNP) sets. The following DYNA3D objects each generate an SNP set:

Symmetry planes with failure
Follower forces
Nodal constraints
Sliding interface definitions 1-10 for Version 2.1
Sliding interface definitions 1, 2, 4, 6, and 7 in Version 4.1
Tie-breaking shell slidelines
Tied node sets with failure
Rigid body joints
Shell-solid interfaces
Discrete springs and dampers
One-dimensional slidelines

Associated with each of the Special Nodal Point sets is a Special Element (SE) set. The SE set consists of all elements that contain one or more nodes in the corresponding SNP set.

All elements in a SE set must be assigned to a single processor. Non-overlapping SE sets can be assigned to different processors, but any one SE set must reside entirely in one processor. This means that the relative size of the largest SE set limits the maximum number of processors that can produce an efficient parallel simulation. For example, if the entire job contains 200000 elements, and the largest SE set contains 20000 elements, then the largest number of processors that the job can be well-partitioned for is ten. If the analyst partitions for more than ten processors, there will be no significant speedup over the ten-processor partition. Since one processor will contain the 20000 elements of the SE set, it will have more computation to do, and the other processors will be forced to periodically wait for the processor handling 20000 elements to complete its calculations.

In most models, if there are large SNP sets, they are due to sliding interfaces. To reduce the size of the largest SNP set, try to reduce the size of the largest sliding interfaces.

3.2 Statistical results in the DynaPart Log file

The statistics for an entire model and for the largest SE sets can be obtained from the log of a DynaPart run. There are several programs that are run by the DynaPart partitioning script, and these programs give job statistics during the partitioning operation. The following statistics for the entire job can be obtained from the screen output from the DynaPart module SNPGEN:

```
Number of materials:                             7
Number of nodal points:                     157331
Number of 8-node hexagonal elements:        142369
Number of 2-node beam elements:               9646
Number of 4-node shell elements:                 0
Number of 8-node thick-shell elements:           0

There were:
        10 Special Nodal Point sets.  The longest set contained
      2208 nodal points, and all sets combined contained a total of
     12690 nodal points.
  There were:
        10 segment sets.  The longest set contained
      2046 segments, and all sets combined contained a total of
     13938 segments.
```

If there are no SNP sets, SNPGEN will tell you, and the job should partition well for any number of processors. In most problems, there are SNP sets, and their size will determine the optimal number of processors for an efficient partition.

If there were one or more SNP sets, then the statistics for the Special Element sets can be determined from the screen output from the DynaPart module AETS:

```
There were:
         4 disjoint sets.  The longest set contained
      3941 elements, and all sets combined contained a total of
```

```
     15764 elements.
```

After the partitioning is completed, the above statistics on the disjoint sets can also be determined by looking at the end of the .dse file. If the name of your job is *bigmodel*, then type the following command at the UNIX prompt:

tail bigmodel.dse

Program REDUCEGRF generates a graph representing the model, and combines all the elements in a Special Element set into a single vertex in the graph. The weight of this vertex represents the computational load for that set. The weight of the largest vertex will determine how many processors the model can be partitioned for while maintaining a good load balance. REDUCEGRF generates output to the user's window that tells the maximum number of processors this job can be partitioned for with a possibility of having a good load balance:

```
Output graph statistics:
  Sum of vertex weights:                      159899
  Largest vertex weight:                        5912
    at reduced vertex:                        136252
    which represents special element set:          1
  Max # partitions for good load balance:         27
```

In this case we partitioned for only 8 processors, so we should get a well load-balanced partition.

The *Computational Load Balance* statistics are obtained from Metis:

```
Edge cuts:  279326,  Load balance:  1.03035,  Score: 1.90380.
```

A balance of 1.00 is a perfect load balance (each processor nominally has the same amount of computation to do at each time step). In this case we obtained a load balance of 1.01, which is a good load balance.

The *Communication Load* statistics are given by DynaPart module PFGEN:

```
                                Number of      Number of
                                   Node         Adjacent
                               Communications   Processors

        Average per processor:      7844.0          4.2
         Max on any processor:     18277            7
      Total on all processors:     62752           34
```

```
At each time step, there are a total of 62752 node communications that must take
place.  The average number of node communications (per processor) is 7844, and
the processor needing to do the most communication needs to communicate 18277
nodes.
```

The performance is best when the number of communications (both the maximum on any processor and the total on all processors) is minimized. This is useful when comparing the efficiency of two different partitions of a model.

MANUAL CHANGE HISTORY

Version 2.1/4.1

February 4, 2004

- Extensive rewrite of the manual to upgrade it from the manual published June, 2000.
- Provided information about the parallel visualization tools, VisIt and EnSight, throughout the manual.
- Discussions on parallel contact algorithms rewritten and updated in Section 2.4 on *Scalable Parallel Contact Algorithms*.
- Provided details on parallel automatic contact algorithms in Section 2.4.2.
- Added Section 2.4.3 and 4.1 on *Modeling Tips for Efficient Parallel Contact*.
- Added Section 2.6 on *Testing and Evaluating Model Scalability.*
- Revised and added paragraphs in Section 3.1 to include the parallel visualization tools.
- Added Section 3.2 to provide path variables for accessing the ParaDyn code and this documentation.
- Rewrote Section 3.3 on *Partitioning a Model*. Added examples illustrating partitioning models with and without contact.
- Updated Section 3.4 to add Mili database names and remove references to Taurus databases.
- Added Section 3.6 on *Running ParaDyn Interactively.* This provides example ParaDyn execution lines for several parallel computers using the utilities **mpirun**, **prun**, **poe**, and **srun**.
- Added Section 3.7 on *Running ParaDyn with Batch* with an example script and pointed to utilities for running under batch systems at the Livermore Computer Center.
- Updated Section 3.8 to focus on Mili databases (Section 3.8.1) and the three graphics post-processors:

  GRIZ4 in Section 3.8.2
  VisIt in Section 3.8.3
  EnSight in Section 3.8.4
- Updated Section 3.9 on *Steps for Running ParaDyn*. Provided a summary of steps in table form that can be printed as a reminder sheet.
- Deleted Section 4.1 on *Static Initialization and Dynamic Analysis***.**
- Added three Appendixes with detailed documentation for DynaPart.
- Added this Manual Change History.